



**Fachhochschule
Bonn-Rhein-Sieg**
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Abschlussarbeit

im Bachelor Studiengang

Entwicklung eines Analysewerkzeugs zur Ermittlung von Metriken und Qualitätskriterien sicherheitsrelevanter Software im Maschinenschutz

von
Christian Staron

Erstbetreuer: Prof. Dr. rer. nat. D. Reinert
Zweitbetreuer: Prof. Dr.-Ing. N. Jung

Eingereicht am: 27. Juli 2004

Inhaltsverzeichnis

Inhaltsverzeichnis	II
Abbildungsverzeichnis	V
Abkürzungsverzeichnis	VII
Abkürzungsverzeichnis	VII
1 Einleitung	1
1.1 Rahmen und Ziele der Arbeit	1
1.2 Aufbau „eingebetteter Systeme“	1
1.3 Einsatz eingebetteter Systeme in Industrie und sicherheitsrelevanten Bereichen	2
1.4 Anforderungen an kritische Systeme und Spezifizierung derer anhand einer Risikoeinstufung	4
1.5 Systemanforderungen sicherheitskritischer Systeme	5
2 Designmethodik und Eigenschaften sicherer und zuverlässiger eingebetteter Systeme	8
2.1 Der Prozess der Softwareentwicklung	8
2.1.1 Vorgehensmodelle der Softwareentwicklung	8
2.1.1.1 Wasserfall – Modell	9
2.1.1.2 V – Modell	9
2.1.1.3 Spiralmodell	10
2.1.2 Grundbegriffe der Softwarequalität	11
2.1.2.1 Beweis	11
2.1.2.2 Verifikation und Validation	11
2.1.2.3 Review, Inspektion, Walkthrough und Schreibtischprüfung	12
2.2 Softwarequalität und deren Sicherstellung im Entwicklungsprozess	13
2.3 Aufbau, Klassifizierung und Entwicklung kritischer Systeme	18
2.4 Software sicherheitsrelevanter Systeme	22
2.4.1 Programmierkonstrukte in der Sicherheitstechnik	23
2.4.1.1 Assembler	23
2.4.1.2 C	24
2.4.1.3 ADA	25
2.5 Merkmale der Hardwarearchitektur kritischer Systeme	25
3 Gewährleistung von Softwarequalität anhand von Metriken	27
3.1 Entwicklung der Metriken	27

3.2	Metriken als Instrument der Qualitätssicherung	28
3.2.1	Automatisierte Softwaremessung	28
3.2.2	Klassifizierung von Softwaremetriken	29
3.2.3	Einordnung der Metriken in Qualitätssicherungsmaßnahmen	30
3.2.4	Gruppierung von Metriken für Softwareprodukte	32
3.2.5	Metriken nach Halstead	35
3.2.6	Metriken nach McCabe	36
3.2.7	Metriken objektorientierter Programmierparadigmen	38
3.3	Anwendung, Ziele und Grenzen der Softwaremessung	38
4	Softwareprüfung und Anforderungen an dabei verwendete Softwarewerkzeuge	40
4.1	Zertifizierung sicherheitsrelevanter Software	40
4.1.1	Das Berufsgenossenschaftliche Institut für Arbeitsschutz	40
4.1.2	Der Zertifizierungsprozess im BIA	40
4.2	Hilfsmittel beim Zertifizierungsprozess – Die Anwendung LOGISCOPE	41
4.3	Evaluierung und Weiterentwicklung der eingesetzten Qualitätssicherungsinstrumente von Krell	45
4.4	Anforderungen seitens des BIA an eine zukünftige Analysesoftware gegenüber dem bisher eingesetzten Werkzeug	46
4.4.1	Die Architektur der Analysesoftware LOGISCOPE	47
4.4.2	Ein exemplarischer Prozess einer Softwareprüfung unter Verwendung von LOGISCOPE	49
4.4.3	Schwachstellen unter Verwendung von LOGISCOPE und Anforderungen an eine zu entwickelnde Rechnerunterstützung	51
5	Softwaredesign der Neuentwicklung	55
5.1	Vorteile einer Eigenentwicklung	55
5.2	Ableitung des Soll – Zustands und dessen Softwarearchitektur aus dem Ist – Prozess	55
5.3	Softwaredesign – Grobentwurf	60
5.3.1	Analyse der Anwendungsfälle und des Gesamtprozesses	60
5.3.2	Auf dem Soll – Konzept aufbauendes Ablaufmodell	62
5.3.3	Komponenten und Subsystemstruktur	62
5.4	Softwaredesign – Feinentwurf und Auswahl der Programmierwerkzeuge	64
5.4.1	Komponenten des Analysewerkzeugs	64
5.4.2	Auswahl der verwendeten Programmierwerkzeuge	65

<u>6</u>	<u>Umsetzung des Softwaredesign – „mEtRIKA“</u>	68
<u>6.1</u>	<u>Microsoft Access, VBA und grundlegende Anwendungsarchitektur</u>	68
<u>6.2</u>	<u>Implementierung der Komponenten des Analysewerkzeugs</u>	71
<u>6.2.1</u>	<u>Graphical User Interface – mEtRIKA</u>	72
<u>6.2.2</u>	<u>Projektverwaltung</u>	74
<u>6.2.3</u>	<u>Verwaltung der Referenzdaten und Berechnung der Kennzahlen</u>	75
<u>6.2.3.1</u>	<u>Generierung der Referenzdaten</u>	76
<u>6.2.3.2</u>	<u>Implementierte Metriken und deren Berechnung</u>	77
<u>6.2.4</u>	<u>Darstellung und Visualisierung der Kennzahlen</u>	83
<u>6.3</u>	<u>Datenhaltung und –verwaltung</u>	85
<u>6.4</u>	<u>Generierung des Auslieferungspakets – mEtRIKA und die Entwicklungsum- gebung auf Verzeichnisebene</u>	87
<u>6.5</u>	<u>Vergleich der mEtRIKA Ergebnisse mit der LOGISCOPE Validierung</u>	88
<u>7</u>	<u>Zusammenfassung und Ausblick</u>	93
	<u>Literaturverzeichnis</u>	VIII
<u>Anhang - 1</u>	<u>In mEtRIKA implementierte Metriken – Übersicht</u>	A1
<u>Anhang - 2</u>	<u>Handbuch – mEtRIKA</u>	B1
<u>Erklärung</u>		XII

Abbildungsverzeichnis

Abbildung 1 – Aufbau eines Mikrocontrollers	2
Abbildung 2 – Die vier Bestandteile der Verlässlichkeit	6
Abbildung 3 – V – Modell	10
Abbildung 4 – Klassifizierung analytischer Qualitätssicherungsverfahren	17
Abbildung 5 – Lebenszyklusmodell nach IEC 61508	20
Abbildung 6 – Software Qualitätsmodell	31
Abbildung 7 – Aspekte der Softwaremetriken	35
Abbildung 8 – McCabe’s zyklomatische Komplexität	37
Abbildung 9 – Beispiel eines Kontrollflussgraphen in LOGISCOPE	42
Abbildung 10 – Beispiel eines Kiviatgraphen in LOGISCOPE	43
Abbildung 11 – Beispiel eines Qualitätsmodells in LOGISCOPE	43
Abbildung 12 – Beispiel eines Aufrufgraphen in LOGISCOPE	44
Abbildung 13 – Verwendete Symbole zur graphischen Visualisierung	46
Abbildung 14 – Architektur LOGISCOPE	47
Abbildung 15 – Auszug einer Referenzkonfiguration in LOGISCOPE	48
Abbildung 16 – Einbettung des Analysewerkzeugs LOGISCOPE in eine Softwareprüfung des Berufsgenossenschaftlichen Instituts für Arbeitsschutz	50
Abbildung 17 – Beispiel einer Ergebnisdatei in LOGISCOPE	51
Abbildung 18 – Soll – Ablaufprozess eines zukünftigen Analysewerkzeugs	58
Abbildung 19 – Softwareverteilungsmuster des zukünftigen Analysewerkzeugs	60
Abbildung 20 – Anwendung des zukünftigen Analysewerkzeugs	61
Abbildung 21 – Komponentendiagramm	64
Abbildung 22 – VBA Entwicklungsumgebung	68
Abbildung 23 – VBA Bibliotheksverweise	69
Abbildung 24 – Microsoft Access GUI	70
Abbildung 25 – Hauptformular mEtRIKA	71
Abbildung 26 – mEtRIKA Modulstruktur	72
Abbildung 27 – SciTE – Editor mit Markierung der Referenzdaten	73
Abbildung 28 – Microsoft Access GUI – gelinkte Tabellen	74
Abbildung 29 – mEtRIKA – Einlesen der Referenzdaten	77
Abbildung 30 – mEtRIKA – Berechnung der Metriken	78
Abbildung 31 – mEtRIKA – Berechnung der Halstead – Operanden und Operatoren	79

Abbildung 32 – mEtRIKA – Berechnung der Halstead – Metriken	80
Abbildung 33 – mEtRIKA – Berechnung der McCabe – Metriken	81
Abbildung 34 – mEtRIKA – Berechnung der Krell – Metriken	82
Abbildung 35 – mEtRIKA – Berechnung der „globalen“ – Metriken	83
Abbildung 36 – mEtRIKA – Beispiel eines digitalen und individualen Berichts	85
Abbildung 37 – mEtRIKA – Backend – Struktur	86
Abbildung 38 – Verzeichnisstruktur zur Generierung eines Installationspakets	87
Abbildung 39 – Untersuchungsgegenstand „ACC TEST“ – Quellcode	88
Abbildung 40 – Untersuchungsgegenstand „ACC TEST“ – KFG	90
Abbildung 41 – Übersicht Testergebnisse mEtRIKA	92

Abkürzungsverzeichnis

ADC	Analog Digital Converter
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
BIA	Berufsgenossenschaftliches Institut für Arbeitsschutz
CASE	Computer Aided Software Engineering
DAC	Digital Analog Converter
DIN	Deutsche Industrie Norm
E/A	Ein- / Ausgabe
Ed.	Editor
EN	Europäische Norm des Comité Européen de Normalisation
EPK	Ereignisgesteuerte Prozessketten
GUI	Graphical User Interface
Hrsg.	Herausgeber
HVBG	Hauptverband der Berufsgenossenschaften
IEC	International Electrotechnical Commission
ISO	Norm der International Organization for Standardization
MTTF	Mean Time to Failure
PDF	Portable Document Format
RAD	Rapid Application Development
RAP	Rapid Application Programming
SIL	Safety Integrity Level
SQL	Structured Query Language
UML	Unified Modelling Language
VB	Visual Basic
VBA	Visual Basic for Application
Y2K	Year 2 Kilo (Synonym für die Jahr 2000 Problematik)

1 Einleitung

1.1 Rahmen und Ziele der Arbeit

Der Studiengang Bachelor of Science in Computer Science der Fachhochschule Bonn-Rhein-Sieg endet mit der Anfertigung einer Bachelor – Abschlussarbeit und dem dazugehörigen Kolloquium. Mit dieser Arbeit galt es ein Analysewerkzeug zu entwickeln, welches im Rahmen einer Softwareprüfung zur Untersuchung sicherheitsrelevanter Software eingesetzt werden kann. Die Anwendung wurde nach den Anforderungen des Berufsgenossenschaftlichen Instituts für Arbeitsschutz (BIA) in Sankt Augustin, welches Softwareprüfungen und -zertifizierungen im Maschinenschutz durchführt, entwickelt. Hauptfunktion der Applikation ist Qualitätskriterien, die Aufschluss über wesentliche Softwareeigenschaften einer zu prüfenden Software liefern, zur Prüfunterstützung zu ermitteln.

In den folgenden Kapiteln wird auf Sicherheitsrelevanz im Zusammenhang mit Rechnersystemen, deren Entwicklung und besonderen Eigenschaften sowie auf den Qualitätsbegriff in diesem Zusammenhang eingegangen. Anschließend wird die Anwendungsentwicklung des Analysewerkzeugs mit Schwerpunkt auf der Designphase und den erzielten Ergebnissen dokumentiert. Aufsetzend auf den Resultaten werden Ausblicke auf weiterführende Funktionalitäten und Erweiterungsansätze betrachtet.

1.2 Aufbau „eingebetteter Systeme“

Der Grossteil der modernen elektronischen Systeme ist mit Rechnern ausgestattet, die insbesondere für diese Art der Anwendung konzipiert und nicht für umfangreiche Gruppen verschiedener Problemstellungen gedacht sind. Derartige Systeme bestehen typischerweise aus einem Chip, auf dem alle weiteren Komponenten angeordnet werden. Ein solches, „eingebettetes System“ (engl.: „embedded system“) setzt auf der Hardwareebene auf und arbeitet mit den angeschlossenen Bauteilen integriert zusammen. In diesem Mikrocontrollersystem sind für die Problemlösung benötigte Komponenten unmittelbar zusammenhängend und an den Prozessor gekoppelt. Entsprechend dieser Betrachtung sind Mehrzwecksysteme (engl.: „general purpose systems“), wie beispielsweise PC – Systeme, für eine Vielzahl verschiedener Anwendungen entworfen und integrieren nicht alle benötigten Bauteile einzig auf einem Chip. Bauteile eines Mikrocontrollers sind zum Beispiel der Prozessor (CPU), Speicher (RAM / ROM), serielle Schnittstellen, Ein- und Ausgabekanäle (E/A – Kanäle), Analog – Digital (ADC) und Digital – Analog – Konverter (DAC) und Sensoren.

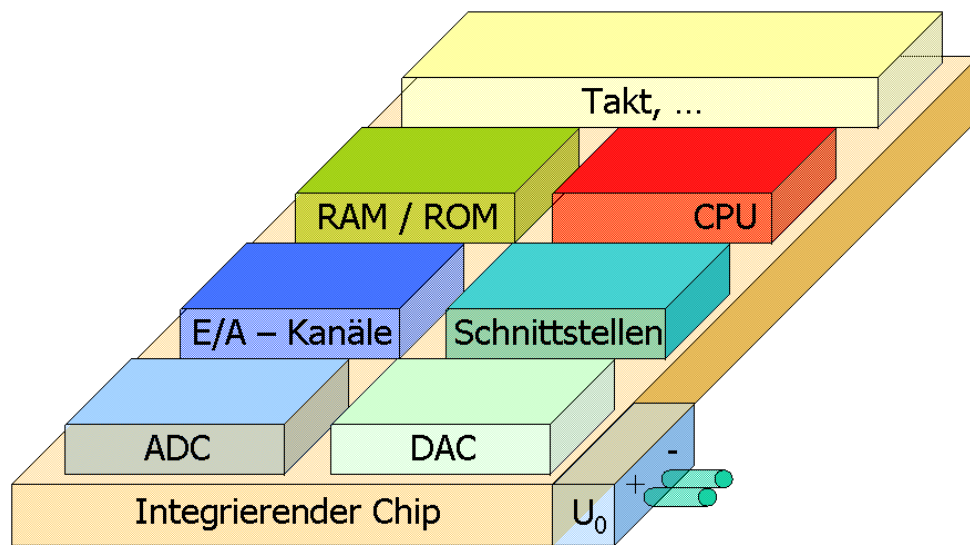


Abbildung 1 – Aufbau eines Mikrocontrollers

[Klug, Schaefer 1997, S. 1, 2]

Derartige Komponenten sind zwar auch häufig in Mehrzwecksystemen vorhanden, doch wie angeführt nicht als „Einchip – Rechner“ integriert. Außerdem lassen sich beide Systeme über die Eigenschaft „Erweiterbarkeit“ voneinander abgrenzen. Mehrzwecksysteme beinhalten einen Prozessor, der auf einer Hauptplatine sitzt und hierüber mit einer Vielzahl verschiedener, auf weiteren Platinen integrierter Systemkomponenten, verknüpft wird. Eingebettete Systeme bieten in ihrer Architektur zwar ebenso Schnittstellen um sie bezüglich ihrer Peripherie zu erweitern, zum Beispiel eine Schnittstelle zu einem externen Speicherbaustein, jedoch nicht in den Ausmaßen, wie sie beim Entwurf von Mehrzwecksystemen vorgesehen werden. Ein Mikrocontrollersystem hat den Anspruch eine bestimmte „Problem – Klasse“ kosteneffizient zu lösen und wird daher nicht für eine Vielzahl denkbarer Problemstellungen eingesetzt. Darüber hinaus lassen sich viele weitere Aspekte bei der Realisierung spezifischer Problemstellungen durch Mikrocontrollerarchitekturen betrachten. Die geringe Größe, der geringe Energieverbrauch oder spezielle Zeitverhalten von Mikrobauteilen sind nur einige in diesem Zusammenhang.

1.3 Einsatz eingebetteter Systeme in Industrie und sicherheitsrelevanten Bereichen

In den letzten Jahrzehnten durchdrang die Elektronik, weite Teile unserer Lebensbereiche. Das Spektrum der Rechnerunterstützung reicht dabei von den „Büroan-

wendungen", wie Textverarbeitung und Audio-, Videoverarbeitung, über Rechenmaschinen zur Entwicklung, Simulation und dem Prototypenbau technischer Einrichtungen bis hin zur Steuerung dieser. Besonders im letzten Bereich sind die typischen Aufgaben von eingebetteten Systemen angesiedelt, da der klassische Maschinenbau häufig nicht mehr alleine tägliche Fertigungsaufgaben in der Industrie bewältigt, sondern dabei verstärkt auf programmierbare Hardware zurückgegriffen wird. Dieser Umstand sorgt für die Synthese beider Fachrichtungen zu einem festen Bestandteil aller Industriezweige und ist als gemeinsame Disziplin immer weniger aus diesen wegzudenken.

Verwendung finden Mikrocontroller in Alltagsgeräten, wie DVD – Playern, Waschmaschinen und Telefonen, sowie in komplexen Systemen, der Automobil-, der Luft- und Raumfahrttechnik, der Anlagensteuerung in Kraftwerken oder der Chemie- und Pharmaindustrie.

Aus Sicht der Anwendungsgebiete betrachtet, lassen sich die unterstützenden Systeme in sicherheitskritische und nicht sicherheitskritische klassifizieren. Ein Merkmal, welches diese Klassifizierung unterstreicht, ist der Schaden, den ein Ausfall der Rechnerunterstützung verursachen könnte. Tritt ein Fehler in einer „Büroanwendung“ oder einem Mikrocontroller eines Telefons auf, ist das ärgerlich, führt aber in der Regel nicht zu einem wesentlichen ökonomischem Schaden für ein Unternehmen, einer ökologischen Katastrophe oder zu Gefahr von Leib und Leben für Personen. Demzufolge bezeichnen wir Systeme, die zur Unterstützung kritischer Anwendungen verwendet werden, wie ein Bremssystem eines Automobils, als sicherheitsrelevante, beziehungsweise -kritische Systeme. Darüber hinaus ist es möglich die Software solcher Systeme weiterführend in zwei Klassen aufzuteilen. Primär sicherheitskritische Software wird direkt in zentralen Steuereinheiten kritischer Systeme eingesetzt. Ihr Ausfall könnte unmittelbar zu einem Schaden führen, wie die zentrale Steuereinheit in unserem Beispiel der Bremsanlage eines Fahrzeugs. Als sekundär kritische Software bezeichnen wir die in der Entwicklung eingesetzten Werkzeuge, die durch ihre eigene Fehlerhaftigkeit ein mangelhaftes Design zur Folge haben könnten und somit indirekte Systemausfälle und damit verbundene Schäden provozieren können. Betrachten wir für unser Beispiel der Bremse einen Ausfall dieses Systems, lässt sich die Verlässlichkeit (Kapitel 1.5) als wesentliche Eigenschaft eines sicherheitskritischen Systems festhalten. [Sommerville 2003, S. 366, 373f]

Die permanent steigende Anwendung von Mikrocontrollersystemen in der Sicherheitstechnik lässt sich durch die hohe Flexibilität eingebetteter Systeme erklären, die es ermöglicht, vielschichtige Aufgaben durch leicht anzupassende Rechnerunterstützung umzusetzen. Immer häufiger werden Mikrocontroller zu Sicherheitszwecken an Maschinen und Anlagen aller Branchen und Industriezweige eingesetzt. [Bömer et al 1998, S. 1; Klug, Schaefer 1997, S. 1, 2; Reinert, Reuß 1991, S. 1] Vor allem die Softwarekomponenten eines eingebetteten Systems ermöglichen, komplexe Systeme, mit beispielsweise einer Vielzahl von Sensoren und Aktoren, über Algorithmen komfortabel zu steuern und zu verwalten. [Sommerville 2003, S. 373f] Dennoch ist nicht jedes System für den Einsatz in der Sicherheitstechnik geeignet. Herkömmliche und bewährte Verfahren können nur durch Rechnerunterstützung erweitert oder abgelöst werden, wenn sie gewährleisten die Aufgabe effizienter oder zuverlässiger zu bewältigen. Ist dies nicht der Fall, spricht kein Argument für den Einsatz einer Rechnerunterstützung. Nach Ehrenberger [Ehrenberger 2002, S. 30] darf in diesem Zusammenhang die Gefahr, die für den Nutzer einer Mikrocontroller – gestützten Anwendung ausgeht, nicht größer sein als die Gefahr, die von der Anwendung ohne Unterstützung ausgeht.

Kommt es zu einem Ausfall einer sicherheitskritischen Anwendung, kann dies zu immensen Kosten führen. Diese Kosten können einerseits direkt mit dem Schaden zusammenhängen, so genannte direkte Ausfallkosten und andererseits indirekt, wie beispielsweise der Imageverlust einer Unternehmung, der zu Geschäftsausfällen führen kann. Aus dieser Notwendigkeit zur Verlässlichkeit des sicherheitskritischen Systems, werden zur Entwicklung in der Regel ausschließlich bewährte Techniken verwendet, zu deren Einsatzverhalten umfangreiche Informationen existieren. [Sommerville 2003, S. 366]

1.4 Anforderungen an kritische Systeme und Spezifizierung derer anhand einer Risikoeinstufung

Nach den vorigen Ausführungen finden sicherheitskritische Systeme häufig Verwendung in Anwendungsbereichen, die prinzipiell keine Ausfälle oder Systemfehler tolerieren können, da diese unmittelbar katastrophale Folgen nach sich ziehen können. Dies impliziert, dass die verwendete Software in einem verantwortlichen System fehlerfrei sein sollte, so dass der Ablauf nicht zu einem Versagen führen kann. Da in der Softwareentwicklung jedoch der Nachweis von Korrektheit und Vollständigkeit bereits bei kleinen Produkten sehr aufwändig bis unmöglich ist, würden solche Pro

zesse die Kosten der Gesamtentwicklung überproportional steigen lassen, so dass das Gesamtprodukt für die Abnehmer zu teuer werden würde.

Um dennoch die Verlässlichkeitsanforderung einer Schutzanwendung bestimmen zu können, wird in einem ersten Schritt der maximale Schaden ermittelt, der durch einen Ausfall verursacht werden könnte. Am Beispiel einer Schutzeinrichtung einer Industriemaschine würde dies bedeuten, dass das von einer Anlage ausgehende Gesamtrisiko bestimmt und anhand von geplanten Schutzeinrichtungen bis mindestens zu einem akzeptablen Restrisiko minimiert wird. Schutzmaßnahmen können in diesem Zusammenhang „nicht – technisch“ sein, wie zum Beispiel Warnhinweise, oder „technisch“ wie Absperrgitter und Sicherheitstechnik in Mikrocontrollern. Eine Maschine wird als „sicher“ eingestuft, wenn das von ihr ausgehende tatsächliche Restrisiko kleiner, beziehungsweise gleich dem vertretbaren Risiko ist. Über diese Methode der Risikoreduzierung werden iterativ die zu installierenden Sicherheitsmechanismen geplant, welche zum Maß für den zu betreibenden Aufwand in der Sicherheitstechnik werden.

Das Risiko beschreibt sich als Funktion aus den Bestandteilen Wahrscheinlichkeit des Eintretens eines Schadens in Verbindung zur Schwere des Schadens. Dabei lässt sich die Schwere des Schadens in verschiedenen Dimensionen ausdrücken und betrachten. Beispiele hierfür sind: Verluste von Menschenleben, Verkürzung der Lebenszeit in Jahren, Tage von Krankenhausaufenthalten, ökologischer Schaden in Quadratkilometern und der Dauer der Verschmutzung oder finanzieller Schaden in Geldeinheiten einer Währung. [Ehrenberger 2002, S. 16f] Je nach sicherheitstechnischer Anwendung können sich Schadenausmaße verschieben, so dass beispielsweise im Maschinenschutz meist ein Schaden den unmittelbaren Verwender der Anlage betrifft. Potentielle Schadenausmaße sind dabei im Bereich von leichten, reversiblen, bis hin zu schweren Verletzungen oder gar Tod des Maschinenführers anzusiedeln.

1.5 Systemanforderungen sicherheitskritischer Systeme

Aus 1.3 wissen wir, dass die Verlässlichkeit ein essentielles Merkmal eines sicherheitskritischen Systems ist. Sie ist mit der Vertrauenswürdigkeit eines Systems gleichzusetzen, welche sich wiederum aus den vier gleichwertigen Bestandteilen Verfügbarkeit, Zuverlässigkeit, Betriebssicherheit und Systemsicherheit zusammensetzt. Um eine Einordnung der Sicherheitsrelevanz aus unserer Sicht in dieses Mo

dell vorzunehmen, werden kurz die Einzelkomponenten der Verlässlichkeit beleuchtet.

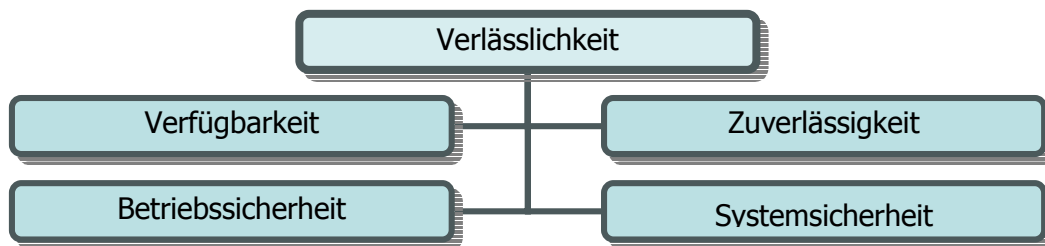


Abbildung 2 – Die vier Bestandteile der Verlässlichkeit

[Sommerville 2003, S. 364]

Jede dieser Eigenschaften ist eine nicht funktionale Anforderung an ein System. Nicht funktional bedeutet in diesem Kontext: „nicht unmittelbar mit den einzelnen Funktionen und deren Korrektheit verknüpft“, sondern vielmehr allgemeine Anforderungen an ein Gesamtsystem. Handelt es sich jedoch bei einer Anforderung um einen zentralen Bestandteil einer Funktion eines Gesamtsystems, darf diese nicht als „nicht funktionale“, sondern muss als funktionale Anforderung verstanden werden.

Die Verfügbarkeit setzt auf die Gesamtlaufzeit eines Systems in der es arbeitet auf, und gewährleistet Dienste in dieser Zeit Anfragenden zur Verfügung zu stellen. Zuverlässigkeit betrachtet die Verfügbarkeit eines Systems in einer festgelegten kürzeren Zeitspanne. Darüber hinaus bedingt die Zuverlässigkeit, dass das umgesetzte System seiner, dem Entwurf zu Grunde liegenden, Spezifikation entspricht, da ein System nur zuverlässig ist, wenn es gemäß den spezifizierten Anforderungen arbeitet. Die Betriebssicherheit (auch: „funktionale Sicherheit“, englisch: „safety“) betrachtet den Risikoaspekt, der von einem System ausgeht, bezüglich der Wahrscheinlichkeit einen ökologischen oder physischen Schaden zu verursachen. Diese Eigenschaft eines Systems beschreibt, wie robust eine Anwendung unter spezifizierten und auch nicht spezifizierten Betriebsbedingungen arbeitet. Aspekte der Informationssicherheit werden über die Systemsicherheit betrachtet. Sie prüft, wie resistent ein System gegenüber Angriffen ist.

Häufig werden die ersten beiden Aspekte der Verlässlichkeit, die Verfügbar- und Zuverlässigkeit als wichtigste Gesichtspunkte angesehen, um die daraus resultierende Verlässlichkeit eines Systems sicherzustellen. Handelt es sich bei der Anwendung um eine sicherheitskritische, wie sie in unserem Kontext zu betrachten ist, spielt das

Merkmal der Betriebssicherheit eine maßgebliche Rolle zur Beschreibung des Sys

Entwicklung eines Analysewerkzeugs zur Ermittlung von Metriken und

Qualitätskriterien sicherheitsrelevanter Software im Maschinenschutz

tems. [Sommerville 2003, S. 111, 363ff, 370f, 373f] In folgenden Ausführungen werden diese Klassifizierungen aufgegriffen und referenziert.

Die Zuverlässigkeit eines Gesamtsystems lässt sich weiter in die jeweiligen Zuverlässigkeiten der dazugehörigen Subkomponenten aufteilen und muss daher auf dieser Ebene betrachtet werden. Ein Ausfall einer Teilkomponente könnte die Eigenschaften peripherer Komponenten beeinträchtigen und einen Ausfall des Gesamtsystems provozieren. Der Begriff der Zuverlässigkeit lässt sich daher in drei Sichten auf ein System einteilen:

- Hardwarezuverlässigkeit: Wie wahrscheinlich ist ein Hardwareausfall?
- Softwarezuverlässigkeit: Wie hoch ist die Wahrscheinlichkeit, dass Softwarekomponenten falsche Ausgaben berechnen und wie robust reagieren die zusammenarbeitenden Algorithmen darauf?
- Bedienzuverlässigkeit: Wie hoch ist die Wahrscheinlichkeit von Benutzerfehlern bei der Bedienung des Gesamtssystems?

Für Hard- und Softwareausfälle von Teilkomponenten ist darüber hinaus zu betrachten, wie robust sich das Gesamtsystem danach verhält und ob es seine Funktionsfähigkeit überhaupt, oder wenigstens eingeschränkt aufrechterhalten kann. [Sommerville 2003, S. 37] Diese Betrachtung der unterschiedlichen Gefahrenquellen eines Systems und seiner Betriebsbedingungen ist Aufgabe der Gefahrenanalyse. Sie soll Gefahren, deren Ursachen und Auswirkungen betrachten und analysieren. Werden die Gefahrenpotentiale verstärkt auf Menschen betrachtet, konzentriert sich die allgemeine Gefahrenanalyse auf eine speziellere Form, die Gefährdungsanalyse. Schritt für Schritt werden dabei die Gefahren und ihre Ursachen ermittelt, die davon ausgehenden Risiken analysiert und daraus resultierend die Gefahren klassifiziert, wonach Maßnahmen zur Risikominimierung, wie auch in Kapitel 1.4 betrachtet, entwickelt werden können. Um die Gefahrenanalyse durchführen zu können, wird häufig die leicht verständliche Fehlerbaumanalyse verwendet. Sie geht vom eintretenden Schaden aus und wird bis auf Bauteilebene heruntergebrochen, um zu erkennen, durch welche Ursachen ein Schaden eintreten kann. [Sommerville 2003, S. 390f]

2 Designmethodik und Eigenschaften sicherer und zuverlässiger eingebetteter Systeme

Der Entwurf sicherheitsrelevanter Systeme stellt wichtige Richtlinien um die in den vorigen Abschnitten beschriebenen Systemanforderungen zu realisieren und wird daher stets in diesem Kontext betrachtet.

2.1 Der Prozess der Softwareentwicklung

Softwareengineering lässt sich durch seine sechs Phasen: Anforderungsanalyse, Spezifikation, Entwurf, Implementierung/Test, Integration/Betrieb und Änderung/Wartung beschreiben. Das Ziel ist eine Software zu entwickeln, deren Design in der Entwurfsphase den Spezifikationen aus der Anforderungsanalyse und Spezifikationsphase entspricht und deren Implementierung wiederum den Entwurf umsetzt. Die Integration der Anwendung in den Betrieb und die Wartung sind Phasen, die nicht deutlich einem Softwareentwicklungsprozess zuzuordnen sind, aber dennoch hinzugehören. Eine Software ist ein lebendiges Objekt, welches zur Einbettung in den Betrieb und zu Wartungsarbeiten immer wieder verändert und angepasst wird. Darüber hinaus zeichnet die Softwareentwicklung weitere Bestandteile aus. Vorgaben und Richtlinien aus der Anforderungsanalyse sind im Produkt zu integrieren und im gesamten Lebenszyklus zu verfolgen und sicherzustellen. Besonders um letzteres zu gewährleisten, stehen dem Gesamtprozess Managementfunktionalitäten zur Hand, die eine permanente Kontrolle der Umsetzung ermöglichen. Somit sind Steuerungsaktivitäten zur Planeinhaltung wesentlicher Bestandteil eines Softwareentwicklungsprozesses und werden weiter in der Qualitätssicherung, dem Kapitel 2.2, vertieft. [Balzert 1998, S. 220; Dumke 1992, S. 14; Ehrenberger 2002, S. 23]

2.1.1 Vorgehensmodelle der Softwareentwicklung

Die so genannten Vorgehens- oder auch Prozessmodelle sind Hilfsmittel zur Abstraktion vom eigentlichen Softwareentwicklungsprozess, um den Überblick über diesen behalten zu können. Sie liefern Werkzeuge die es ermöglichen, Einzelschritte zu vereinfachen und zu beschreiben, um benötigte Ressourcen gezielter und effizienter im Gesamtprozess einsetzen zu können. Ressourcen sind in diesem Fall Mitarbeiter und ihre Kompetenzen, Entwicklungswerkzeuge, Methoden, Räumlichkeiten und Finanzen. Der Rahmen, der durch ein Modell gesteckt wird, ist vom durchführenden Projektmanagement variierbar und dient dazu, Teilprozesse in ihrem zeitlichen Ablauf und ihren Ergebnissen zu fixieren. Die Einzelabläufe liefern

definierte Teilprodukte die im Verlauf der Gesamtentwicklung zusammengesetzt und daher integriert werden. [Balzert 1998, S. 98; Sommerville 2003, S. 24]

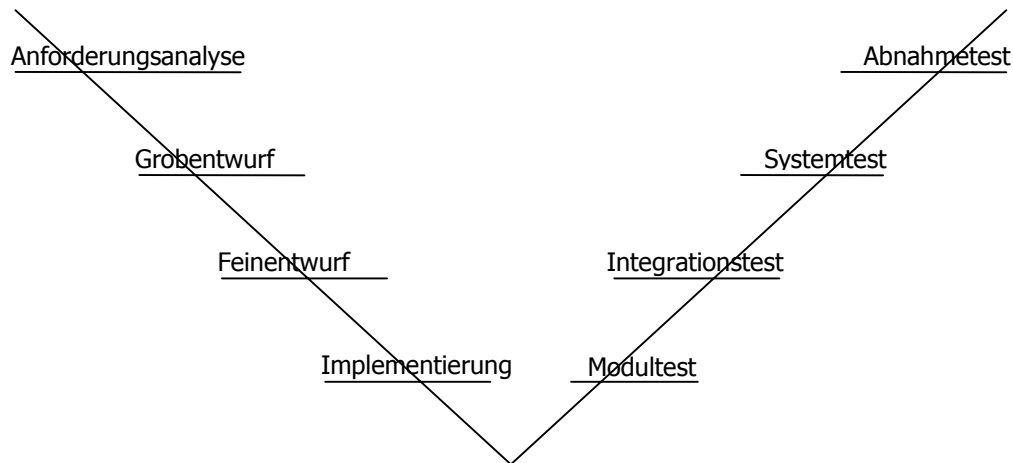
Zur Erläuterung werden in folgenden Abschnitten die drei Prozessmodelle Wasserfallmodell, V – Modell und Spiralmodell aufgeführt, um einen Überblick über häufig im Softwareengineering verwendete Verfahren und Methoden aufzuzeigen. Weitere Vorgehensmodelle, wie das evolutionäre/inkrementelle Modell, das Prototypenmodell, nebenläufige und objektorientierte Modelle oder Extreme Programming werden an dieser Stelle nicht weiter betrachtet.

2.1.1.1 Wasserfall – Modell

Ähnlich einem Wasserfall verläuft dabei der Softwareengineering – Prozess in Stufen abfallend, wobei die Produkte und Ergebnisse einer vorigen Stufe zur Grundlage der folgenden werden. Es gibt einen definierten und sukzessiven Verlauf des Gesamtmodells durch alle in 2.1 betrachteten Phasen des Gesamtprozesses. Nach der Theorie sind alle Dokumente und Ergebnisse einer Phase von den Beteiligten „abzunehmen“, bevor sie als Grundlage für den Folgeprozess dienen. Sind Änderungen im Entwurf durchzuführen, die durch Probleme in einer späteren Stufe auftreten, können iterativ Rückführungsschleifen verwendet werden, die jedoch lediglich einen Rücksprung zum unmittelbar vorigen Prozess ermöglichen. Nach einer Rückführung müssen alle folgenden Phasen des Modells wieder sukzessive durchlaufen werden. [Balzert 1998, S. 98; Sommerville 2003, S. 24, 57f; Vogel-Heuser 2003, S. 25f]

2.1.1.2 V – Modell

Könnte man von einer Philosophie des V – Modells sprechen, würde es der Qualitätsanspruch an ein zu entwickelndes Produkt sein. Gemäß dem Ziel „safe to market“ ist das Modell eine konsequente Fortführung des Wasserfallmodells, mit den entsprechenden Qualitätssicherungsmaßnahmen der einzelnen Phasen auf der, den Entwicklungsphasen gegenüberliegenden Seite.

**Abbildung 3 – V – Modell**

[In Anlehnung an Balzert 1998, S. 101]

Verlaufen die Entwicklungsphasen dem obigen Modell entsprechend abfallend, ist jeder dieser ein Qualitätssicherungsprozess für die in der Phase entstandenen Dokumente und Teilprodukte zugeordnet. Dadurch können Fehler und Kontroversen zu den spezifizierten Anforderungen in den Teilprodukten erkannt und korrigiert werden, bevor sie in das Gesamtsystem eingehen und zu Ausfällen führen können. Nach der Literatur eignet sich das Modell besonders für große Projekte und Entwicklungen sicherheitskritischer Anwendungen, da es einen fest definierten Rahmen für Qualitätssicherung, Änderungs- und Konfigurationsmanagement enthält. Nachteilig wird das Modell, wenn es für kleine und mittelgroße Softwareentwicklungen verwendet wird. Durch seinen strengen und festen Rahmen führt es in diesem Fall zu einer unnötigen „Softwarebürokratie“ und würde den Entwicklungsprozess daher eher behindern als fördern. [Balzert 1998, S. 98, 101, 113, 135; Vogel-Heuser 2003, S. 25ff]

2.1.1.3 Spiralmodell

Das Spiralmodell spaltet den Softwareentwicklungsprozess in vier Teile auf, die immer wieder zyklisch im Gesamtprozess durchlaufen werden. Die vier Bereiche sind: 1. Ermitteln von Zielen, Alternativen und Randbedingungen, 2. die Fortentwicklung der Alternativen und Identifikation von Risiken und deren Verringerung, 3. Entwicklung und Qualitätssicherung des Produkts, 4. Planung der nächsten Phase. [Sommerville 2003, S. 65ff; Vogel-Heuser 2003, S. 27]

2.1.2 Grundbegriffe der Softwarequalität

Im Zusammenhang zur Qualitätssicherung im Softwareentwicklungsprozess und deren Maßnahmen treten häufig Begriffe auf, die einer Erklärung bedürfen, um sie voneinander abgrenzen und gezielt in folgenden Teilen der Arbeit einsetzen zu können.

2.1.2.1 Beweis

Ein Beweis liefert als Ergebnis für eine Behauptung, die vorher aufgestellt und durch den Beweis formal bestätigt oder widerlegt wird, eine logische Aussage wie „wahr“ oder „falsch“. Die formale Herleitung eines korrekten Resultats ist nur bei einer genau spezifizierten Behauptung möglich. In unserem Zusammenhang erfordert dies, dass eine Spezifikation, hier die Behauptung, formal notiert ist, um Missverständnisse, die beispielsweise durch die Beschreibung anhand natürlicher Sprache entstehen könnten, als fehlerhaftes Fundament des Beweises zu vermeiden. [Ehrenberger 2002, S. 19] Besonders in Verbindung zur Qualitätssicherung bei Algorithmen liefern Beweise zwar genaue Aussagen, sind jedoch sehr aufwändig und im Hinblick auf Entwicklungskosten nur bedingt durchführbar.

2.1.2.2 Verifikation und Validation

Ebenso unter den Begriffen Verifizierung und Validierung bekannt, sind sie Methoden für die Überprüfung der Übereinstimmung zwischen den spezifizierten Anforderungen einer Softwareentwicklung und dem Softwaredesign in der Entwurfsphase sowie mit der letztendlichen Implementierung einer Anwendung. Daher werden sie als eine Art der Qualitätssicherung und Qualitätskontrolle im Kontext des Engineering – Prozesses von Beginn bis zum Ende des Produktlebenszyklus verwendet. Beispielsweise werden sie häufig als Qualitätssicherungsmaßnahmen im V – Modell (2.1.1.2) eingesetzt, um Fehler in Teilkomponenten aufzuspüren und die Übereinstimmung der Spezifikation mit der Umsetzung zu wahren. [Consolini 2001, S. 34; Paradiso 2001, S. 36; Sommerville 2003, S. 427] Um beide Begriffe zu erläutern werden sie jetzt in getrennten Zusammenhängen betrachtet.

Die Verifikation integriert den formalen Nachweis (2.1.2.1) in dem Prozess der Qualitätssicherung und ist eine Methode, um die Übereinstimmung zwischen Spezifikation und Implementierung für alle in einer Anwendung vorkommenden

Eingaben nachzuweisen. Ebenfalls wird sie häufig dazu verwendet, Inkonsistenzen zwischen verschiedenen Dokumenten, beispielsweise aus Grob- und Feinentwurf aufzudecken und so frühzeitig Designfehler zu finden. Sie stellt eine Methode im Qualitätssicherungsprozess dar, um Aussagen über die Korrektheit und Vollständigkeit einer Implementierung nachzuweisen. Durch die dafür benötigte Formalität entsprechend dem Beweis ist sie jedoch sehr aufwändig und daher kostenintensiv. [Balzert 1998, S.101, 446; Ehrenberger 2002, S. 19; Vogel-Heuser 2003, S. 26]

Validierung beschreibt dagegen eine kostengünstigere Form der Qualitätssicherung. Sie vermittelt zwar keine allgemeingültigen Aussagen, wie sie die Verifikation leistet, kann jedoch für den Grossteil der zu prüfenden Anwendungen hinreichende Aussagen bezüglich der qualitativen Eignung treffen. Aufgrund von spezifizierten Szenarien und „gut“ gewählten Testmengen, mit Standard- und Grenzwerten, sowie unzulässigen Eingaben, kann die Funktionsweise nach spezifizierten Anforderungen kostengünstig und effizient ermittelt werden. Darüber hinaus dient die Validierung häufig zur Prüfung einzelner Dokumente im Kontext des Gesamtsystems, was die Kompatibilität von gefertigten Einzelkomponenten bei ihrer Zusammenführung gewährleisten soll. [Balzert 1998, S. 101; Ehrenberger 2002, S. 20; Vogel-Heuser 2003, S. 26]

2.1.2.3 Review, Inspektion, Walkthrough und Schreibtischprüfung

Diese vier Begriffe sind ebenfalls Bestandteile eines typischen Softwareengineeringprozesses und werden daher auch in diesem Kontext erläutert. In der Praxis werden sie häufig als Verfahren zur Validation und Verifikation von Prüflingen verwendet, um gemäß den Erläuterungen aus den obigen Abschnitten die Übereinstimmung zwischen Spezifikationen und Umsetzungen zu prüfen.

Reviews sind immer in Verbindung zum Gesamtprojekt und weniger bezüglich eines einzelnen Softwareprodukts zu betrachten. Sie haben den Anspruch den Soll – Ist – Zustand eines Projekts zu prüfen und gegebenenfalls den zukünftigen Projektfortgang zu steuern und zu beeinflussen. Bisherige Prozesse werden besprochen und ausgewertet und gehen in die zukünftige Planung ein. [Balzert 1998, S. 317; Ehrenberger 2002, S. 171f]

Die Inspektion, der Walkthrough und die Schreibtischprüfung beziehen sich im Gegensatz zum Review auf einzelne Softwareprodukte aus den Implementie

rungsphasen und stellen verschiedene Arten der Qualitätskontrolle und Qualitätssicherung dar, um Fehler, Inkonsistenzen und Unvollständigkeiten aufzudecken. Bereits in dieser Reihenfolge der Nennung ist eine qualitative Abschwächung, aufgrund der Beteiligten und der Durchführungsart zwischen den einzelnen Methoden festzustellen.

Grundlage einer Inspektion kann jegliches, in einem Entwicklungsprozess erarbeitete Dokument sein. Anforderungen, Spezifikationen, Grob- und Feinentwürfe, Diagramme, Programmcodes, Testresultate, Abnahmeberichte, Projektpläne und viele weitere Dokumente können während des gesamten Prozesses von den Besprechungsmitgliedern in verteilten Rollen nach Vorbereitung einer Sitzung inspiziert und begutachtet werden. [Balzert 1998, S. 305; Ehrenberger 2002, S. 171f]

Der Walkthrough wird dagegen von dem Autor des Dokuments ohne eine klare Rollenverteilung bei den sonstigen Teammitgliedern geleitet. Ebenfalls hat in dieser Sitzungsform der Verfasser die Aufgabe die Beteiligten durch das Dokument zu führen. Das Verfahren gilt als qualitativ schlechter, da der Autor durch seine Resultate führt und somit bereits eine „eingefahrene“ Sichtweise auf Problemstellungen beim Durchgang vorgegeben ist. [Balzert 1998, S. 321; Ehrenberger 2002, S. 171, 176]

Bei der Schreibtischprüfung existiert keine formalisierte Methode. Ein Gutachter prüft die Dokumente für sich allein, um ebenfalls Fehler und die Produktqualität festzustellen. Typischerweise ist die Schreibtischprüfung durch den Autor weitaus weniger wirkungsvoll als durch einen unabhängigen Gutachter, der die Dokumente prüft. [Ehrenberger 2002, S. 171, 176f]

2.2 Softwarequalität und deren Sicherstellung im Entwicklungsprozess

Der Begriff der Softwarequalität kann aus vielerlei Perspektiven betrachtet und daher nicht eindeutig zugeordnet werden. Verschiedene Sichten auf die Qualität eines Produkts bilden die vier Ansätze der Transzendenz, des Produkt-, Benutzer- und Prozessbezugs. [Balzert 1998, S. 256]

- Der transzendente (auch „übernatürliche“) Ansatz sagt aus, dass die Qualität eines Produkts immer erkennbar und für jeden Betrachter unmittelbar einsichtig ist. Dieser Ansatz der Bewertung von Produktqualität ist jedoch für die Praxis nicht anwendbar.

- Produktbezug meint, den objektiven Blick auf ein zu untersuchendes Softwareprodukt ohne sich von subjektiven Wahrnehmungen manipulieren lassen zu können. Ein Produkt ist dabei genau spezifizier- und quantifizierbar, so dass gemessene Werte eine Aussage und einen Vergleichswert gegenüber anderen Produkten liefern. Der Aspekt des Kunden, der einen Nutzen aus der Verwendung des Produkts erkennen muss, wird hierbei außer Acht gelassen.
- Der benutzerbezogene Ansatz sieht den Kunden als einzigen vor, der die Qualität eines Produkts beurteilen kann. Dabei stellt die Subjektivität des Benutzers das größte Problem dar.
- Der Prozessbezug sagt aus, dass die Produktqualität ausschließlich durch die korrekte Herstellung eines Produkts gewährleistet wird und die Spezifikation und Kontrolle des Herstellungsprozess dabei die Qualität als Kosten/Nutzen – Funktion beeinflusst.

In der Realität lässt sich die Qualität einer Software durch eine Mischung der letzten drei Ansätze beschreiben. Ein Produkt muss anhand von Maßstäben in einer gewissen Weise quantifizierbar sein, wie beispielsweise Quelltexte aufgrund ihrer Struktur oder die Anzahl der Funktionen, die sie dem Benutzer auf der Benutzerschnittstelle zur Verfügung stellen. Der Benutzer entscheidet sich wiederum für ein Produkt und wählt es nach seinen Maßstäben der Produktqualität aus. Anforderungen, nach denen er ein Produkt bewertet, entsprechen häufig den nicht Funktionalen aus Kapitel 1.5, über die er die wesentlichen Charakteristika einer Applikation beschreibt.

Produktionsprozesse beeinflussen die Softwareentwicklung und -qualität. Wird nachlässig bei der Entwicklung gearbeitet, entstehen hohe Kosten durch benötigte Nachbesserungen in Wartungsphasen. [Balzert 1998, S. 21f]

Aus den vorigen Kapiteln wissen wir, dass Qualitätssicherungsmaßnahmen in Vorgehensmodellen integriert sein können. Alle Qualitätssicherungsprozesse werden dem Qualitätsmanagement zugeordnet, welches während einer Entwicklung anhand von Techniken die geplante Softwarequalität gewährleisten soll. Teilaufgaben des Managements sind das festlegen von Zielen und Verantwortungen zur Verfolgung der planmäßigen Qualitätspolitik. Die Qualitätsplanung, -lenkung, -sicherung und Qualitätsverbesserung im Umkreis des Managementsystems sind dabei unterstützende Hilfsmittel. Eine Definition des Qualitätsmanagements mit einer vollständigen Liste der dazugehörigen Tätigkeiten liefert die Norm DIN EN ISO 8402. [Balzert 1998, S. 278] Gemäß den Herleitungen von Qualitätsmerkmalen für sicherheitskritische Sys

teme aus Kapitel 1.4 und 1.5, spezifiziert die Norm DIN ISO 9126 diese und weiterführende Qualitätseigenschaften von Softwareprodukten. Weitere, dabei berücksichtigte Attribute sind [Balzert 2000, S. 1102f]:

- Funktionalität unter Berücksichtigung der Richtigkeit, Angemessenheit, Interoperabilität, Ordnungsmäßigkeit und Sicherheit
- Zuverlässigkeit, Reife, Stabilität, Fehlertoleranz und Wiederherstellbarkeit von betroffenen Daten und des Leistungsniveaus unter Betrachtung der benötigten Zeit und des Aufwands
- Benutzbarkeit
- Effizienz unter Betrachtung des Zeit- und Verbrauchsverhaltens von Ressourcen
- Änderbar-, Wartbar- und Modifizierbarkeit bezüglich Aufwand von Korrekturen, Verbesserungen oder Anpassungen unter Berücksichtigung der Analysier- und Prüfbarkeit
- Übertragbar- und Portierbarkeit zwischen Systemplattformen und Systemumgebungen

Die Ergebnisse der einzelnen Parameter beschreiben die Qualität der Software und ermöglichen eine Klassifizierung derer in vorher festgelegte Anforderungen und Grenzwerte. [Balzert 1998, S. 257]

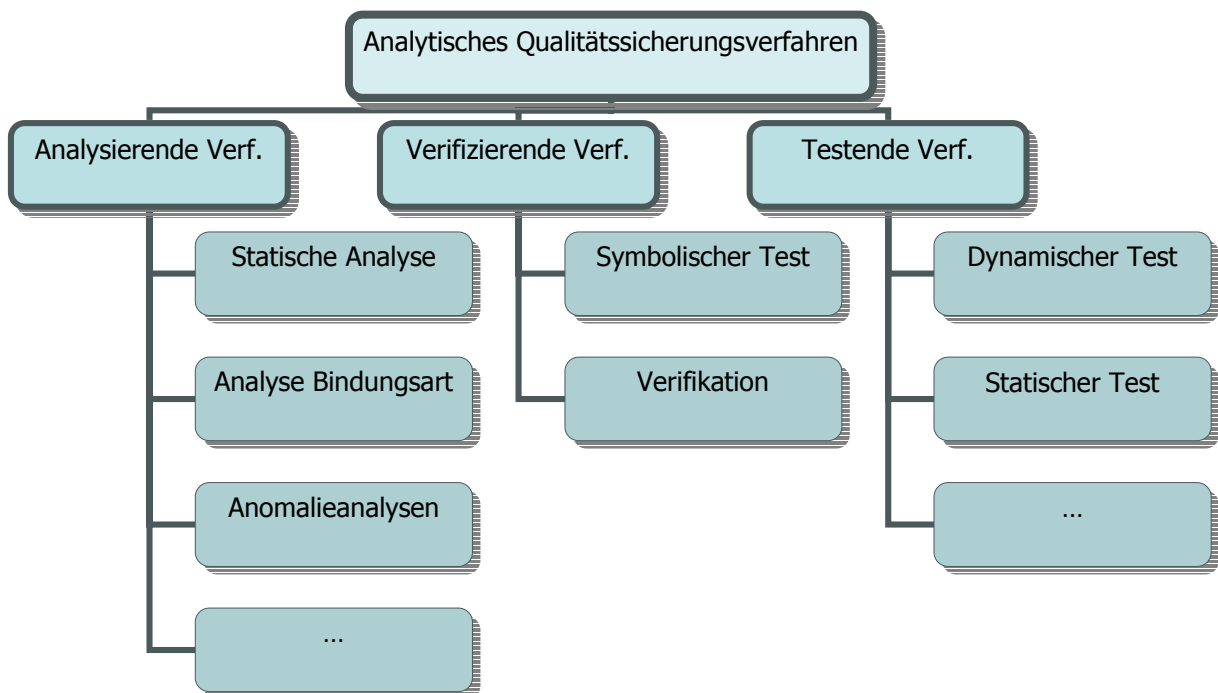
Um Softwarequalität zu prüfen stehen den Verantwortlichen eine Reihe von Verfahren und Methoden zur Verfügung. Informelle Qualitätssicherungsmaßnahmen, wie sie in Kapitel 2.1.2.3 beschrieben werden, können im vollständigen Entwicklungszeitraum eingesetzt werden, führen aber gegenüber dem allgemeinen Nachweis der Verifikation aus Kapitel 2.1.2.2 nicht immer zu eindeutigen und vollständigen Resultaten. Häufig liefern sie jedoch frühzeitig Indikatoren für Probleme und Fehler, da sie bereits während der Entwicklung als Prüfverfahren für Teilprodukte der einzelnen Entwicklungsphasen eingesetzt werden können. Als wichtiger Aspekt gilt dabei, dass die Kosten für die Beseitigung eines Fehlers in der Entwicklungsphase um ein vielfaches geringer sind, als wenn der Fehler erst nach der Implementierung entdeckt wird. [Ehrenberger 2002, S. 167]

Verfahren mit eindeutigen Aussagen sind:

- Testende Verfahren: Dabei wird eine zu prüfende Software mit definierten Eingaben (Testmengen) ausgeführt und das Verhalten des Prüflings dokumentiert. Vorteil dieser dynamischen Testverfahren ist, dass das Produkt in seiner realen Umgebung mit Daten getestet wird und daraufhin eindeutige Aussagen über

die Funktionalität mit die Testdaten gemacht werden können. Statische Testverfahren betrachten den Analysegegenstand nicht während seines Ablaufs. Dabei wird meist der Quellcode direkt überprüft um Fehler festzustellen. [Balzert 1998, S. 280f, 395f]

- Verifizierende Verfahren: Wie bei der Beschreibung der Verifikation in 2.1.2.2 erläutert, versucht sie den formalen Nachweis der Übereinstimmung eines Produkts mit dessen Spezifikation. Bei dieser Form der Auswertung wird die Software mit symbolischen Ausdrücken, ergo Eingabevariablen, interpretiert und getestet. Ähnlich wie bei den testenden Verfahren, nur dass die Einschränkung durch die Testmenge entfällt, um allgemeine Aussagen erzielen zu können. [Balzert 1998, S. 395f; DGQ 1992, S. 100f]
- Analysierende Verfahren: Auch analytische Verfahren genannt, untersuchen sie statisch ein zu prüfendes Softwareprodukt. Die Prüfung basiert auf den quantifizierbaren Eigenschaften, wie beispielsweise der Struktur des Quellcodes. Die analysierenden Verfahren können fortgehend in die Arten: Analyse der Bindungsarten von Systemkomponenten, Metriken und Anomalienanalysen klassifiziert werden. Dabei liefern erstere Ergebnisse über den Zusammenhang zwischen einzelnen Systemmodulen. Die Metriken beschreiben die Untersuchung des Quellcodes, wie nach dem vorig erläuterten Beispiel der Programmstrukturanalyse, basierend auf den Quelltexten. Die Anomalienanalysen ermöglichen systematische Analysen von Abweichungen, wie beispielsweise Datenflussanomalien. [Balzert 1998, S. 280f; 395f]

**Abbildung 4 – Klassifizierung analytischer Qualitätssicherungsverfahren**

[In Anlehnung an Balzert 1998, S. 397]

Alle Verfahren lassen sich nach ihrem Automatisierungsgrad strukturieren. Automatisierung ist durch Werkzeuge und Verfahren zu erzielen, die die vollständigen oder Teilprozesse automatisiert abarbeiten können, so dass nicht alle Prüfungen der Produktqualität manuell durchgeführt werden müssen. Typischerweise ist es bisher so, dass semantische Prüfungen manuell vorgenommen werden müssen, wobei für die syntaktischen, Konsistenz- und Vollständigkeitsprüfungen unterstützend automatisierende Werkzeuge existieren. [Balzert 1998, S. 302]

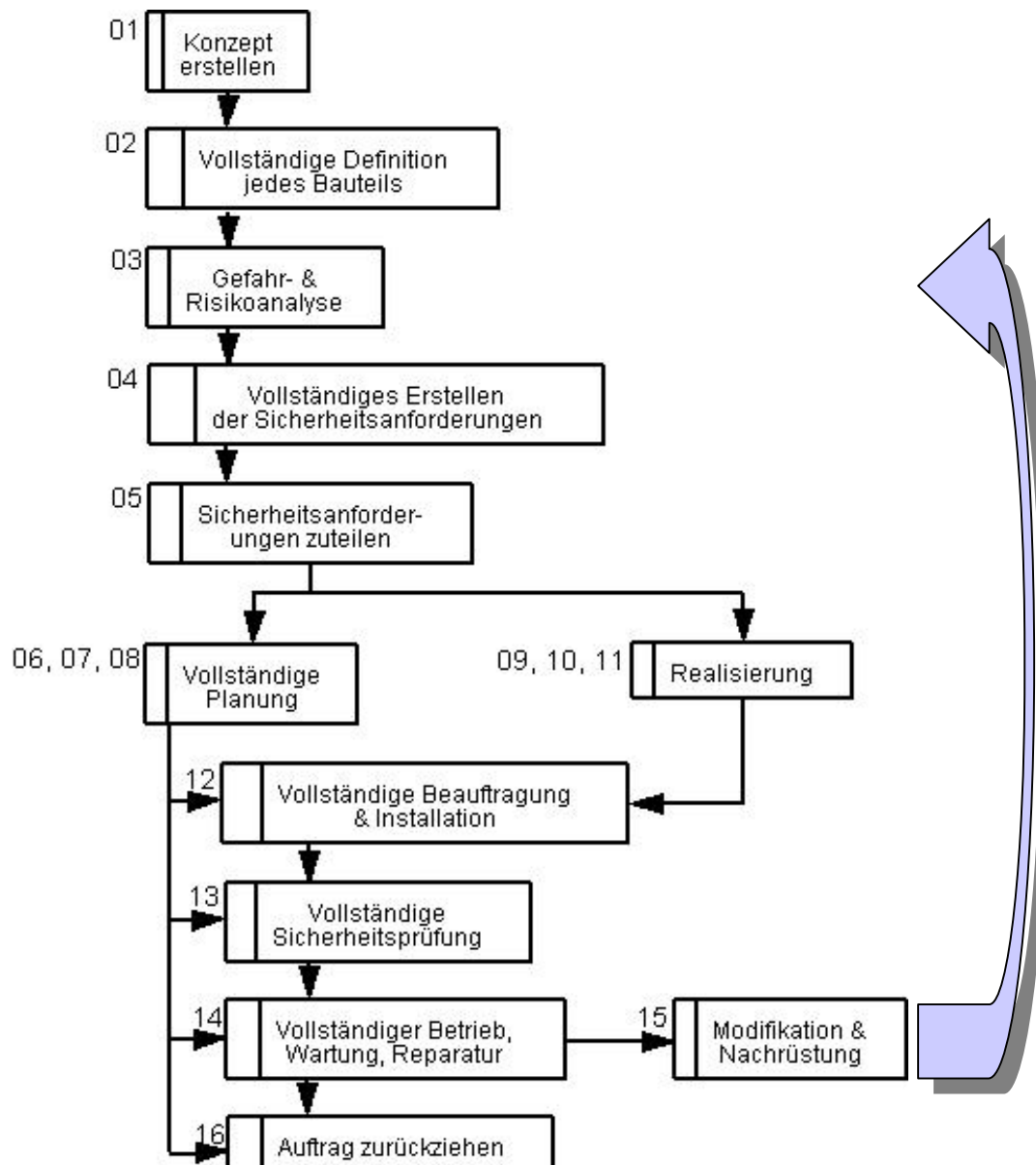
2.3 Aufbau, Klassifizierung und Entwicklung kritischer Systeme

Wie bereits in Kapitel 1.3 erläutert, werden kritische Systeme häufig für die Steuerung von Maschinen und Anlagen verwendet. Eine Anforderung, die aus vielen Anwendungsgebieten resultiert, ist die Echtzeitfähigkeit, was bedeutet, dass Systeme innerhalb eines definierten Zeitraums auf Ereignisse reagieren. Häufig belaufen sich die Zeitintervalle in der Praxis auf wenige Mikrosekunden. Oft lassen sich Echtzeitsysteme als Zweistufensysteme durch ein Eingangsereignis (auch „Stimulus“), welches auf das System wirkt, und eine Antwort („Response“) des Systems auf den Eingang klassifizieren. Weiter lassen sie sich über die Eigenschaften der Stimuli gruppieren. Periodische Eingangsereignisse treten in einem definierten Zeitfenster immer wieder auf. Aperiodische Ereignisse treten in unregelmäßigen Abständen auf und sind somit für ein System nicht vorhersagbar. Diese Eingänge werden üblicherweise anhand eines so genannten „Interrupt – Handler“ abgearbeitet. Tritt ein Ereignis auf, wird eine „Unterbrechung“ an das System weitergegeben, welches darauf nach definierten Prioritäten reagieren kann. [Sommerville 2003, S. 293f]

Die Betriebssicherheit konnte in Kapitel 1.5 als wesentliche Eigenschaft sicherheitskritischer Systeme herausgestellt werden. Jede Gefahr, die durch den Ausfall einer Anwendung droht, muss ausgehend von ihrem potentiellen Risiko des Eintretens eingestuft werden. Die Klassifizierung nach der Norm der International Electrotechnical Commission (IEC) 61508 ermöglicht eine solche Einstufung und dient als Maß zur Integration von Sicherheitsmaßnahmen in ein Gesamtsystem. Über diese Einordnung der Gefahr lässt sich ableiten, welcher der vier Risikoreduzierungsklassen das System genügen muss. Ebenfalls existieren, sich durch die Einstufung eines Softwareprodukts in den „Safety Integrity Level“ (SIL) unterscheidende, Anforderungen an die Softwarequalität und den Softwareentwicklungsprozess. Anschließend lassen sich daraus Systemverhalten im Gefahrfall und Präventivmaßnahmen spezifi

zieren und entwickeln. [Ehrenberger 2002, S. 26; Reinert 2001, S. 14; Reinert, Reuß 1991, S. 1; Vogel-Heuser 2003, S. 30f]

Weiterhin unterscheidet die Norm nach der IEC in fehlervermeidende und fehlerbeherrschende Maßnahmen. Sie gelten als Erweiterungen des klassischen Softwareengineering und dienen einem besseren Qualitätsmanagement sowie aus Sicht der definierten Qualitätsparameter einem besseren Softwareprodukt.

**Abbildung 5 – Lebenszyklusmodell nach IEC 61508**

[In Anlehnung an Vogel-Heuser 2003, S. 31]

Häufig ist die Ursache von Systemausfällen nicht auf Bauteilausfälle aufgrund beispielsweise Komponententalterung zurückzuführen. Vielmehr gelten nicht berücksichtigte und spezifizierte Systemzustände als häufige Ausfallursachen, da sie zu einem nicht definierten Systemverhalten führen. Fehlervermeidende Maßnahmen finden im vollständigen Produktlebenszyklus Anwendung, da sie bereits in der Entwicklung beginnen und bis zum Ende der Produktverwendung andauern. Fehlervermeidung entspricht der Strategie, in der Produktentwicklung bewährte Techniken, die beispielsweise die Wahrscheinlichkeit von Programmierfehlern senken oder be

währte Validationsmethoden darstellen, zu verwenden sowie eine strikte Qualitätspolitik zu verfolgen. Qualitätsmanagementtechniken finden permanent zur Sicherung des Qualitätsplans Anwendung. Jegliche Fehler, die im Design und der Konzeption gemacht werden können, gilt es durch Anwendung der Strategien vorzubeugen und zu vermeiden. Die sicherheitstechnische Entwicklung findet analog der herkömmlichen Produktentwicklung statt. Da Softwarefehler nur in der Erstentwicklung beziehungsweise durch Änderungen in das Produkt gelangen können, gelten auch bei Wartungsarbeiten die Prinzipien der Fehlervermeidung. Die in Kapitel 2.2 aufgezeigten Methoden zur Sicherstellung der Softwarequalität sind demnach der Fehlervermeidung im Softwareengineeringprozess zuzuordnen.

Fehlerbeherrschende Maßnahmen finden, im Gegensatz zu den Fehlervermeidenden, erst im Produkt selbst Anwendung, müssen jedoch ebenso in den Entwicklungsphasen vorgesehen und integriert werden. Sie gehen von Fehlern im Produkt aus, sind daher als fehlertolerant einzustufen, und versuchen diese anhand sicherheitstechnischen Mechanismen zu kompensieren. Über Fehlererkennung und die Integration von Redundanzen auf Systemebene wird versucht, Fehlern und Entwicklungsmängeln in Software entgegenzutreten. Beherrschung meint in diesem Sinne die präventive Erkennung von Fehlern um den Systemausfall zu verhindern. Maßnahmen sind zum Beispiel Selbsttests des Systems, bei denen ausführbare Befehle oder Hardwaremodule wie Speicher, Register und Ein-, Ausgabeeinheiten hinsichtlich ihrer korrekten Verarbeitung und Funktionsweise geprüft werden. [Klug, Schaefer 1997, S. 4ff; Sommerville 2003, S. 401f; Reinert 2001, S. 12, 16; Reinert, Reuß 1991, S. 1f] Fehlerbeherrschende Mechanismen können durch ihren zusätzlichen Aufwand, den sie im Entwicklungsprozess verursachen, den Teilbereichen Verfügbarkeit und Zuverlässigkeit, der in 1.5 spezifizierten Verlässlichkeit sicherheitsrelevanter Systeme zugeordnet werden. In diesem Zusammenhang berühren sie die zwei Eigenschaften, da ein Ausfall eines fehlerbeherrschenden Mechanismus im definierten Zeitraum den geforderten Dienst nicht erbringen könnte und die Verlässlichkeit des Gesamtsystems gefährdet.

Die Entwicklung sicherheitsrelevanter Systeme erfordert daher ein besonderes Maß an Qualitätsmechanismen, die die spezifizierten und erforderlichen Anforderungen gewährleisten. In, für diese Form der Entwicklung angewandten Vorgehensmodellen sind demnach die Phasen der Risikoanalyse, Spezifikation, Entwurf und Implemen

tierung von Sicherheitsmechanismen sowie deren Integration in das Gesamtprodukt des Systems und die Qualitätssicherungsmaßnahmen hinzuzufügen. Unterstützung bieten so genannte Computer Aided Software Engineering (CASE) – Werkzeuge, die den Einsatz graphischer Methoden und Modelle im gesamten Entwicklungsprozess ermöglichen. Hilfreich sind diese Verfahren, da sie durch Konsistenzprüfungen zwischen den einzelnen Entwicklungsphasen die Vollständigkeit und Korrektheit der einzelnen Produkte kontrollieren und Abweichungen, die zu Fehler führen könnten, aufdecken. [Reinert, Reuß 1991, S. 5f; Vogel-Heuser 2003, S. 30f]

2.4 Software sicherheitsrelevanter Systeme

Software gibt die Verarbeitungsschritte eines programmierbaren Systems vor und ist daher, ähnlich der Hardware, wesentlicher Bestandteil eines solchen Systems. Der Entwicklungsprozess sicherheitskritischer Software findet unter besonderen Betrachtungen und Anforderungen bezüglich der Qualität und Korrektheit statt. Daher sind dabei eine Reihe von Konventionen zu berücksichtigen, die die Systeme bezüglich der in der Norm DIN ISO 9126 spezifizierten Attribute stabilisieren sollen. Beispielsweise sollten sicherheitskritische Elemente von eben nicht kritischen in der Software gekapselt werden um eine Vermischung beider Bereiche zu vermeiden und Wartungsarbeiten zu erleichtern. Im Programmierstil ist zu beachten, dass sicherheitsrelevante Systemkomponenten so verständlich, wie nur möglich entwickelt werden. Möglichst trickreiche, komplizierte und „knapp“ formulierte Programmteile sind vielleicht für den ersten Autor einsichtig, können jedoch bei einer Wartung oder Anpassung zu einem späteren Zeitpunkt zu Schwierigkeiten, oder gar Fehlern führen und somit zu Ausfallursachen werden. Außerdem ist die Verwendung von Programmierkonstrukten und –verfahren die unnötig Fehler provozieren könnten oder von besonderen Spracheigenschaften, die nicht klar definiert sind, nicht zu empfehlen. Beispiele hierfür sind „goto“ – Anweisungen, Fließkommazahlen und die so genannten „Zeiger“ der Programmiersprache C (Kapitel 2.4.1.2), deren Verwendung in einer Ebene > 2 aufgrund der Komplexität leicht zu Missverständnissen führen können. Prinzipiell ist es wichtig auf die Lesbarkeit von Programmen zu achten um anderen Mitarbeitern den Einstieg in diese nicht zu erschweren. [Sommerville 2003, S. 401f, 421f]

2.4.1 Programmierkonstrukte in der Sicherheitstechnik

Eine, für eine sicherheitsrelevante Anwendung verwendete Programmiersprache liefert mit ihren Programmierkonstrukten und Datentypen wichtige Eigenschaften für die Anwendung und Entwicklung. Nach Auswahl einer Programmiersprache machen sich deren Eigenschaften unmittelbar in der Entwurfsphase bemerkbar. Das Design, die verwendeten Werkzeuge und nicht zuletzt die Funktionen der Anwendung werden durch die gewählte Sprache beeinflusst und können sie beschneiden oder erweitern. Die Spracheigenschaften geben Aufschluss darüber, was eine Anwendung ermöglichen kann und wo ihre Grenzen sind. Häufig muss bei der Wahl zwischen den Parametern Ausführungsgeschwindigkeit, Übersichtlichkeit und Testbarkeit ausgewogen werden. [Bömer et al 1998, S. 1; Sommerville 2003, S. 298, 401f] Folgend werden drei häufig verwendete Programmiersprachen im Zusammenhang mit Mikrocontrollersystemen erläutert und an diesen beispielhaft aufgezeigt, welche Parameter bei der Wahl eines Softwareprodukts beachtet werden müssen.

2.4.1.1 Assembler

Jeder, auf einem Mikrocontroller eingesetzte Prozessor, hat einen eigenen Befehlssatz, den interpretieren und ausführen kann. Dieses Vokabular bezeichnet man als Assemblerdialekt oder Assemblersprache. Er dient der Programmierung des Prozessors. Nach der eigentlichen Maschinensprache, die auf Platinebene abläuft und aus binären Zahlenkolonnen besteht, ist Assembler die erste Möglichkeit einen Prozessor abstrahiert von der Maschinensprache und für einen Menschen verständlich zu programmieren. Die Befehle eines Prozessors lassen sich in die Kategorien arithmetische und logische Befehle, Bitmanipulations-, Vergleichs-, Schiebe- und Rotationsbefehle, Datentransfer-, Sprungbefehle und Befehle zum Unterprogrammaufruf, zur Stackmanipulation sowie sonstige Steuerbefehle und Sonderbefehle aufteilen. Assemblerprogramme haben die Nachteile, dass sie nur schwer auf andere Plattformen portierbar sind, da die verwendeten Befehle dem Befehlssatz des Prozessors entsprechen müssen. Da Assemblerprogramme schwer lesbar und meistens nicht intuitiv verstehbar sind, müssen sich Programmierer und Prüfer von solchen Programmen gut in den Befehlssatz und die Software einarbeiten um den Ablauf und die Korrektheit nachvollziehen zu können. Beispielsweise müssen einfache Schleifenkonstrukte aus Hochsprachen auf einzelne Befehle abgebildet und Variablen auf Registerebene

betrachtet werden, was die Lesbarkeit von Assemblerdialekten stark beeinträchtigt. Dennoch werden je nach Anwendung auch heute noch Assemblerprogramme den Programmen in Hochsprachen vorgezogen. Durch die Hardwarenähe zeichnet sie eine sehr kurze Ausführungszeit und keine Einschränkungen der Prozessorfunktionen aus, die gewöhnlich bei Hochsprachen durch eine Abstraktion vom Prozessor zustande kommen können. Bislang finden sie oft in Echtzeit- und Sicherheitsanwendungen ihren Einsatz. Um die Vorteile beider Programmierparadigmen zu nutzen werden auch Hoch- und Assemblersprachen im so genannten „Inline“ – Assembler verknüpft. Dabei werden zeitkritische Assemblermodule in Hochsprachencodes integriert und als Assembler ausgeführt. [Bömer et al 1998, S. 315 – 321; Sommerville 2003, S. 298]

2.4.1.2 C

Die Programmiersprache C gilt im Gegensatz zu den Assemblerdialekten als Hochsprache und wurde durch die Anwendung in der Betriebssystemprogrammierung seit 1972 und die Standardisierung im Jahre 1982 zum ANSI C populär. Ihre Vorteile sind die Hardwarenähe, die zwar nicht mit Assemblersprachen vergleichbar, dennoch aber effizienter als bei anderen Hochsprachen ist, sowie ihre gute Lesbar- und Zuverlässigkeit, die durch die Standardisierung ermöglicht wurde. Durch die Optimierung auf die basierende Hardware ermöglicht sie eine schnelle Verarbeitung und verknüpft diese mit den Vorteilen von Hochsprachen: leicht verständliche Programmierkonstrukte, Datentypen und Portabilität. Wesentlicher Nachteil bildet in C die Speicherverwaltung, bei der Ressourcen manuell reserviert und freigegeben werden können, die aber bei nicht sachgemäßer Verwendung, zum vollständigen Systemabsturz führen können. Außerdem lädt die Programmiersprache dazu ein, Entwicklungsschwerpunkte auf die Ausführungsgeschwindigkeit zu legen, so dass die Gefahr besteht, Programme schnell „kryptisch“ und schlecht wartbar zu entwickeln. Daher sind Übereinkünfte im Programmierstil bei C – Entwicklungen unerlässlich für die Transparenz der Quellcodes. Dabei sollte ebenso, besonders im Hinblick auf sicherheitskritische Anwendungen, von der unnötigen Verwendung von zu Fehlern führenden Methoden abgesehen werden. Die Anwendung von Zeigern in der „x-ten“ Ebene, eröffnet beispielsweise elegante, aber häufig schwer durchschaubare und daher potentiell gefährliche Lösungsmöglichkeiten. [Bömer et al 1998, S. 233f, 240-244; Sommerville 2003, S. 298]

2.4.1.3 ADA

Eine speziell zur Unterstützung maschinennaher Anwendungen, wie Steuerungen und weitreichende Automatisierungen, entwickelte Programmiersprache ist ADA. Sie entstand aus der explosionsartigen Nachfrage nach eingebetteten Anwendungen die immer komplexere und umfangreichere Aufgaben übernehmen sollten. Als direkter Nachfahre der Programmiersprache PASCAL übernimmt sie große Teile daraus und ergänzt die Aspekte der Ausnahmebehandlungen, Parallelverarbeitung von Prozessen und der dazugehörigen Synchronisation, sowie Echtzeitkontrolle. [Bömer et al 1998, S. 149f, 154; Sommerville 2003, S. 298]

2.5 Merkmale der Hardwarearchitektur kritischer Systeme

Eingebettete Systeme liegen häufig sicherheitskritischen Anwendungen zugrunde, so dass die Architektur durch einen, alle Komponenten integrierenden Chip beschrieben wird. Diese Rechner werden nach ihrer Anwendung einem Safety Integrity Level (SIL) der Norm IEC 61508 zugeordnet, der sicherheitstechnische Anforderungen, ausgehend vom potentiellen Schaden durch einen Systemausfall, definiert. Dieses so festgelegte Sicherheitsniveau ist bei der Entwicklung des Gesamtsystems zu berücksichtigen und schlägt sich unmittelbar in fehlervermeidenden und fehlerbeherrschenden Maßnahmen des Systems und somit im Engineeringprozess und in der Hardwarearchitektur nieder. Durch den vorgegebenen Maßnahmenkatalog können sich Ansprüche ergeben, die mehrkanalige Strukturen erfordern um beim Ausfall eines Moduls über Redundanzen arbeiten zu können. Darauf aufsetzend ist der kreuzweise Vergleich übertragener Daten auf redundanten Kanälen eine verschärfende Maßnahme. Wie man bereits an diesem Beispiel sieht, sind die Anforderungen unmittelbar an Hardwareressourcen geknüpft, so dass sie von Beginn an in der Entwicklung einer Anwendung berücksichtigt und einbezogen werden müssen. Aber nicht nur direkter Hardwareaufwand muss in die Konzeption eines kritischen Systems integriert werden. Sicherheitskritische Systeme erfordern, je nach der SIL – Einstufung, weitere fehlerbeherrschende Maßnahmen auf Softwareebene, die sich auf die Hardwareressourcen niederschlagen und daher deren leistungsstärkere Dimensionierung erfordern. Ein Beispiel dafür sind Softwarebasierte Systemtests, wie Befehlstests oder Tests verschiedener Hardwarebauteile, die während der normalen Systemverarbeitung ausgeführt werden müssen und daher bis zu 50% der Hardwareressourcen beanspruchen können. Die entsprechende Einplanung der

artiger Aufwendungen ist demnach in der Konzeption einer Hardwarearchitektur eines kritischen Systems unerlässlich.

Darüber hinaus werden softwarebasierte fehlerbeherrschende Mechanismen über wiederum zu entwickelnde Softwarekonzepte umgesetzt, was deren Prüfung über fehlervermeidende Maßnahmen erfordert. Außerdem gilt es über die Software fehlerbeherrschende Mechanismen in sicherheitsrelevanten Applikationen zu integrieren, wodurch die Grundstruktur der eigentlichen Applikationslogik häufig verändert werden muss, so dass eine Analyse der Gesamtlogik oftmals besondere Fehlererkennungsstrukturen aufweist. [Klug, Schaefer 1997, S. 1, 4ff, 8f; Reinert, Reuß 1991, S. 2]

3 Gewährleistung von Softwarequalität anhand von Metriken

3.1 Entwicklung der Metriken

In vielen Naturwissenschaften ist Messen ein wichtiges Verfahren um die Funktionsweise von Theorien und Methoden zu belegen und zu untermauern. [Dumke 1992, S. 33f] Wie in Kapitel 2.2 erläutert, gehören analytische Verfahren zu den gängigen Methoden die Qualität einer Software einzuschätzen und zu beurteilen. Die Sicht auf das Produkt und seinen Entwicklungsprozess lassen viele Ansätze der Qualitätsprüfung zu.

Historisch betrachtet wurde die Messung von Hardwarebauteileigenschaften vor der Messung von Softwareattributen entwickelt und angewandt um Komponentenzuverlässigkeit zu überprüfen. Eine bis heute weit verbreitete Hardwaremetrik ist „Mean Time to Failure“ (MTTF), die eine Aussage darüber trifft, wie lange die durchschnittliche Funktionsdauer eines Hardwarebauteils ist. [Sommerville 2003, S. 383]

Seit Mitte der 70er Jahre steigt die Bedeutung der Softwaremessung im Softwareentwicklungsprozess stetig an, so dass sie eine gängige Methode geworden ist Umfang und Funktion von Softwareprodukten zu quantifizieren und zu messen. In den 80er und 90er Jahren wurden die Softwaremetriken um die Messung einzelner Softwarekomponenten, ergo kleineren Teilen des Gesamtprodukts, und um die Messung des Entwicklungsprozesses erweitert, um deren Effizienz und Qualität zu steigern, Kosten und die Fehlerwahrscheinlichkeit im Endprodukt zu senken. Heute hat die Softwaremessung einen wichtigen Stellenwert in Softwareengineering – Prozessen eingenommen. Softwarestruktur, deren Modularität und Einfachheit, sowie die Effizienz der Prozesse wurden nunmehr als wesentliche Kriterien für betriebssichere und somit sicherheitsrelevante Software (Kapitel 1.5) erkannt und daher als primär zu erreichende Ziele angestrebt. Metriken eines Softwareprodukts liefern Indikatoren und Kennzahlen, die Aussagen über seine Beschaffenheit treffen und somit ein wichtiges Instrument der Qualitätsprüfung darstellen. Sie helfen produktspezifische Prozesse zu verstehen, Kontrolle über diese Prozesse zu bewahren und Verbesserungsansätze für sie entwickeln zu können. [Balzert 1998, S. 225f; Dumke 1992, S. VII, 35; Orci 2001, S. 71; Fenton, Pfleeger 1997, S. 3, 13; Zuse 1999, S. 3]

3.2 Metriken als Instrument der Qualitätssicherung

Softwaremessungen setzen bei dem zu entwickelnden Gesamtprodukt, Teilprodukten, bei deren Entwicklungsprozessen, oder bei darin verwendeten Entwurfs- und Programmiersprachen, Methoden oder Werkzeugen auf und sind daher vielseitig einsetzbar. Dabei ist das Ziel anhand von numerischen Werten, Produkte mit Standards bezüglich Programmierrichtlinien oder Produkte untereinander, vergleichbar zu machen und somit Aussagen über die Qualität der Prüflinge treffen zu können. Da die Prüfungen bereits auf, während der Engineeringphasen bis dato entwickelten Teilprodukten und Dokumenten aufsetzen können, liefern sie frühe Indikatoren um die Gesamtentwicklung entsprechend der definierten Ziele lenken zu können. [Dumke 1992, S. 3, 8, 39; Jacobsen-Rey 2000, S. 1f; Sommerville 2003, S. 557ff]

Gutachter erhalten über Metriken einen Eindruck zur Charakteristik eines Softwareprodukts und somit einen schnellen Einstieg in die Prüfung. Sie erkennen anhand von Messergebnissen, wann die spezifizierten Anforderungen bezüglich Programmierrichtlinien und Standards erfüllt sind und die Entwicklung konsistent mit der Spezifikation und dem Entwurf ist. Ebenso ist es für Kunden während der Entwicklung, wie vor der Abnahme ein Instrument, zur Überprüfung der in Auftrag gegebenen Software, deren Eigenschaften und Qualität. Muss die Software gewartet werden ermöglichen Messresultate den Mitarbeitern einen Eindruck über die Problemstellen und „Hot Spots“ der Anwendung. An diesen auffälligen Komponenten können sie dann mit den Verbesserungsarbeiten aufsetzen. [Fenton, Pfleeger 1997, S. 3; Jacobsen-Rey 2000, S. 1, 10; Sommerville 2003, S. 560]

3.2.1 Automatisierte Softwaremessung

Inspektionen, wie die analysierenden Verfahren aus 2.2, lassen sich manuell, und inzwischen zu großen Teilen automatisiert durchführen. Diese Methoden ermöglichen die einfache Erstellung von Softwaremetriken und somit die Unterstützung des Qualitätsmanagements.

Ein Beispiel für den Vorteil der automatisierten Unterstützung von Softwaremessungen ist das Jahr 2000 – Problem (Y2K := engl. „year 2 kilo“), welches damals nicht ausschließlich ein technisches, sondern vielmehr ein organisatorisches Problem, aufgrund der hohen Verbreitung von kritischen Anwendungen, darstellte. Anhand von speziell für die Problematik entworfenen Softwarewerkzeugen, den sogenannten „Y2K Independent Verification & Validation“ – Anwendungen, konnte in jeglichen Institutionen und Organisationen eingesetzte Software auf Inkonsisten

zen und Fehler bezüglich der Jahr 2000 Problematik hin automatisiert untersucht werden. Ebenso wie in dem Beispiel, können unterstützende Werkzeuge Softwareprüflinge bezüglich anderer spezifizierter Problematiken hin untersuchen und die Resultate Gutachtern und Prüfern unterstützend bereitstellen. Kein dabei verwendetes Werkzeug ist ein Garant um Fehler in Prüflingen aufzudecken. Vielmehr ist es eine Technik, um einen ersten Einblick in komplexe Softwaresysteme zu erhalten, und um auf kritische und daher zu prüfende Punkte hingewiesen zu werden. Eine automatisierte Codeinspektion kann während des vollständigen Produktlebenszyklus eingesetzt werden um Qualitätsmetriken zu erzeugen und diese in Berichten zusammengefasst und aufbereitet dem Qualitätsmanagement zur Verfügung zu stellen. Wie jedoch häufig in der Praxis stellt sich die Mischung der manuellen und automatisierten Inspektionen als effektivste Methode zur Fehlervermeidung und zur Erreichung der besten Softwarequalität heraus. [Dumke 1992, S. 173; Dumke, Winkler 1999, S. 173-193; Jacobsen-Rey 2000, S. 1f, 3f, 10]

3.2.2 Klassifizierung von Softwaremetriken

Metriken lassen sich durch ihre Untersuchungsgegenstände klassifizieren und strukturieren. Die größten Gebiete der Softwaremessung betreffen die Analyse des eigentlichen Produkts, dessen Entwicklungsprozesse und wiederum deren Ressourcen.

- **Softwareressourcen:** Der für einen Entwicklungsprozess benötigte Input sind die Ressourcen, die für diesen bereitgestellt werden. Dies können beispielsweise Verfahren, Werkzeuge, Standards und Designregeln, Hardware, Geschäftsräume und Personal sein.
- **Softwareprozess:** Die Metriken zur Untersuchung eines Softwareprozesses analysieren Aktivitäten und Teilprozesse, die mit der Softwareentwicklung als solche im Zusammenhang stehen. Dabei zu quantifizierende Ansatzpunkte sind zum Beispiel die Erstellung der Spezifikation und des Anforderungskatalogs, der Grob- und Feinentwurf der Software und Anwendung findende Testmethoden.
- **Softwareprodukt:** Das Produkt steht als Ergebnis des Softwareprozesses in der Hierarchie als letzter Punkt. Betrachtete und untersuchte Einheiten sind bei dieser Analyse die Produkte der einzelnen Softwareprozesse, wie beispielsweise Dokumente, Codeelemente und Testergebnisse.

Wird in Verknüpfung zu den Softwareprodukten der Fokus auf zu analysierende Codekomponenten gelegt, spielen Programmstruktur, verwendete Programmiersprache, Entwicklungsumgebung und weitere verwendete Werkzeuge, Testtechniken und das Programmierungsmanagement eine maßgebliche Rolle. Programmierungsmanagement beleuchtet dabei Aspekte wie beispielsweise die Bestückung von Entwicklungsteams. Dabei werden auch bei separat betrachteten Codemodulen alle drei Segmente der Softwareanalyse: Ressourcen, Prozesse und Produkte involviert.

Für die Untersuchungen in dieser Arbeit sind die Metriken bezüglich des Softwareprodukts maßgeblich und werden daher fortlaufend fokussiert. Die drei Bereiche der Softwareanalyse können zudem in interne und externe Attribute klassifiziert werden. Interne Eigenschaften sind diejenigen, die unmittelbar aus dem untersuchten Objekt abgeleitet und ermittelt werden können. Die externen haben demzufolge nicht die direkte Verknüpfung zum analysierten Objekt. Die Umgebung und das Verhalten des Objekts in dieser, sind dabei maßgebliche Untersuchungsgegenstände. [Dumke 1992, S. 11, 47f; Fenton, Pfleeger 1997, S. 74ff; Orzi 2001, S. 74]

3.2.3 Einordnung der Metriken in Qualitätssicherungsmaßnahmen

Um weiterführend die Qualität von Software bestimmen zu können, müssen die Eigenschaften, die eine Software nach ihrer Spezifikation erfüllen soll festgelegt werden. Die nach DIN ISO 9126 definierten und in Kapitel 2.2 beschriebenen Softwareattribute können von einem Produkt erfüllt werden. Um dies zu prüfen, werden einem Prüfling zugeordnete Qualitätsmerkmale wiederum in einzelne Faktoren aufgespalten, die gemeinsam das Qualitätsmerkmal ergeben. So würde beispielsweise das Qualitätskriterium Wartbarkeit durch die einzelnen Faktoren Analysierbarkeit, Änderbarkeit, Testbarkeit und Stabilität erfüllt werden. Um nun eine Software bezüglich ihrer Wartbarkeit untersuchen zu können, muss geprüft werden, ob die Software eben den einzelnen Faktoren Analysierbarkeit, Änderbarkeit, Testbarkeit und Stabilität genügen kann. Dafür wird wiederum jeder der einzelnen Faktoren in einzelne Metriken aufgespalten, diese gemeinsam betrachtet bestimmen, wie gut beispielsweise ein Prüfling zu analysieren ist. Demnach existiert ein Sortiment an Metriken, die in Kombination einzelne Faktoren darstellen. Die Faktoren wiederum gruppiert bilden ein Qualitätskriterium der Software. Um bestimmen zu können wie gut oder schlecht ein Softwareprodukt einem Faktor

entspricht, werden Ober- und Untergrenzen für die Metriken vergeben, die beeinflussen, ob eine Teilmetrik in die Wertung eingeht. Fällt eine Metrik zum Beispiel „schlecht“ aus, über- oder unterschreitet sie einen gesetzten Grenzwert und geht nicht anteilig in die Gesamtwertung ein. Durch diesen Wegfall könnte ein Faktor, wie zum Beispiel die Testbarkeit niemals 100% erlangen, da eine dazugehörige Teilmetrik weggefallen ist. Auf diesem Weg ist es möglich, Aussagen darüber zu treffen, in wie fern ein Prüfling einen Faktor einer Eigenschaft und daraus ableitend die Eigenschaft selbst erfüllt oder nicht erfüllt. Ein dabei häufig verwendetes Qualitätsmodell ist von McCall, der darin einzelne Qualitätsmerkmale verschiedenen Produktfunktionalitäten zuweist. [Balzert 1998, S. 262; Dumke 1992, S. 6; Fenton, Pfleeger 1997, S. 16f, 18; Sommerville 2003, S. 557ff; Windpassinger 2000, S. 137ff]

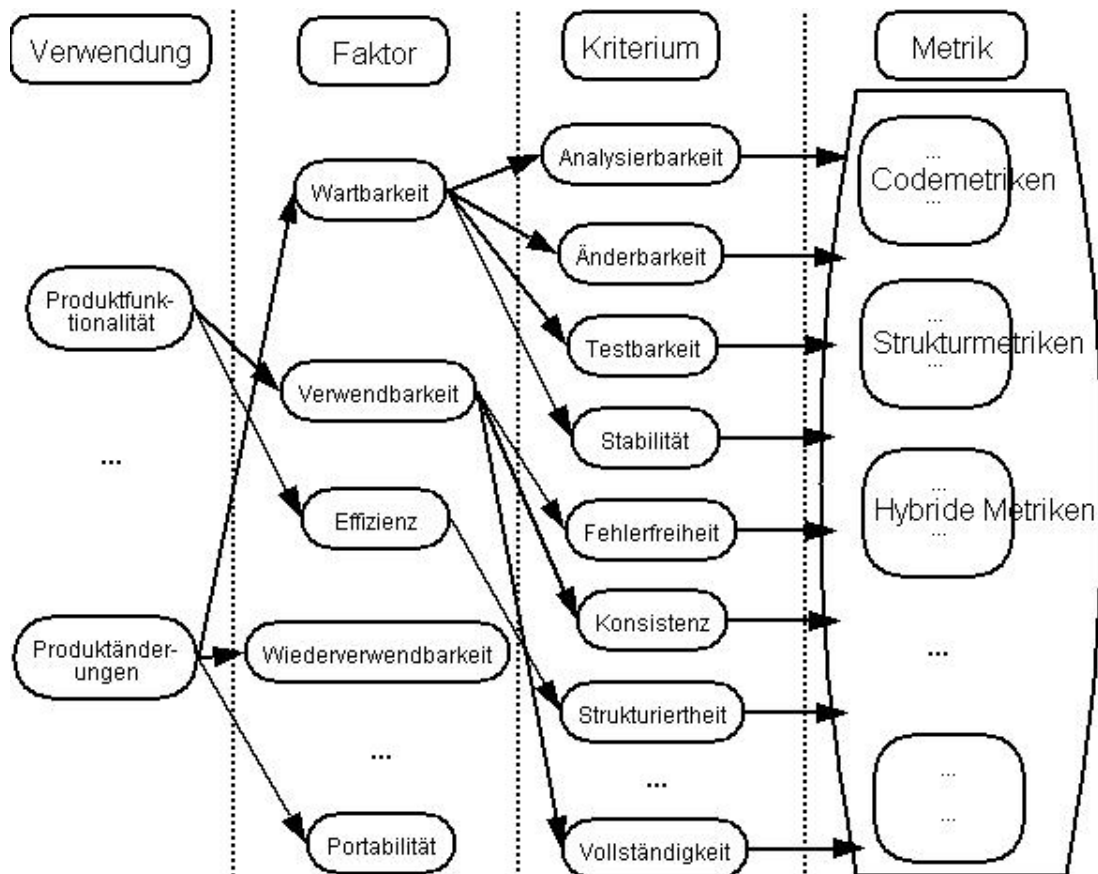


Abbildung 6 – Software Qualitätsmodell

[In Anlehnung an Dumke, Lehner 2000, S. 138; Fenton, Pfleeger 1997, S. 17]

3.2.4 Gruppierung von Metriken für Softwareprodukte

Aufgrund des zugrunde liegenden Messprinzips lassen sich Produktmetriken in statische und dynamische Metriken aufteilen. Eine dynamische Messung findet während des Programmablaufs statt, im Gegensatz zu einer statischen, die nur die Quelltextstruktur analysiert. Nach Sommerville [Sommerville 2003, S. 561] helfen statische Messungen bei der Beurteilung von Komplexität und Wartungseigenschaften eines Systems und nicht, wie die dynamischen Messungen, bei der Beurteilung der Leistungsfähigkeit und Zuverlässigkeit. Die statisch erstellten Metriken können in die Kategorien Codemetriken, Strukturmetriken und hybride Metriken eingeteilt werden. Erstere sind auch als linguistische Metriken bekannt. Sie messen die syntaktischen Einheiten eines Quellcodes und geben quantitative Aussagen über den Prüfling, wie zum Beispiel über den Umfang des Programms. Häufig erwähnte Metriken sind in diesem Zusammenhang die Metriken nach Halstead, der mit seinen Veröffentlichungen Mitte der 70er Jahre Vorreiter auf diesem Gebiet war. Der Sektor der strukturellen Metriken wurde, ebenso Mitte der 70er Jahre, von McCabe erschlossen. Er ist Vater der zyklomatischen Komplexität, welche in der zyklomatischen Zahl ausgedrückt wird. Auch bekannt als logische Strukturmetriken setzen sie auf den Kontrollfluss innerhalb eines Programms auf und beschreiben in welcher Sequenz Programminstruktionen abgearbeitet werden. Durch diese Art der Untersuchung werden Schleifenstrukturen und daraus resultierend der Programmfluss eines Produkts beschrieben. Graphisch können die Kontrollflussstrukturen als gerichtete Graphen (Kontrollflussgraphen) dargestellt werden. Hybride Metriken, also diese, die beide Aspekte der linguistischen Spracheigenschaften einer verwendeten Programmiersprache und die logische Struktur eines Programms betrachten, wurden von Henry und Kafura entwickelt. Ihre Informationsflussmetriken kombinieren die Strukturmetriken und die Messungen zum syntaktischen Aufbau.

Daneben existieren Metriken, die an dieser Stelle keine wesentliche Rolle spielen und daher lediglich kurz aufgegriffen werden. Zu ihnen gehören Datenstrukturmetriken, die die Lebenszeit und Referenzen von, im Programm verwendeten Variablen analysieren und so genannte Stilmetriken, die Quelltexte auf den verwendeten Programmierstil bezüglich Einrücktiefe, etc. untersuchen. Außerdem werden Ansätze zur Prüfung weiterer Programmkomplexitäten, wie der Problemkomplexität und der rechenbetonten Komplexität angewandt. Erstere beziehen sich auf die Untersuchung des, einem Problem zu Grunde liegenden Algorithmus. Der zweite

Ansatz beschäftigt sich mit der algorithmischen Komplexität, die Aussagen darüber trifft, wie effizient ein Problem anhand von Logik gelöst wurde. [Balzert 1998, S. 479; Dumke 1992, S. VII; Edwards et al 1999, S. 197; Fenton, Pfleeger 1997, S. 243-245, 279-282]

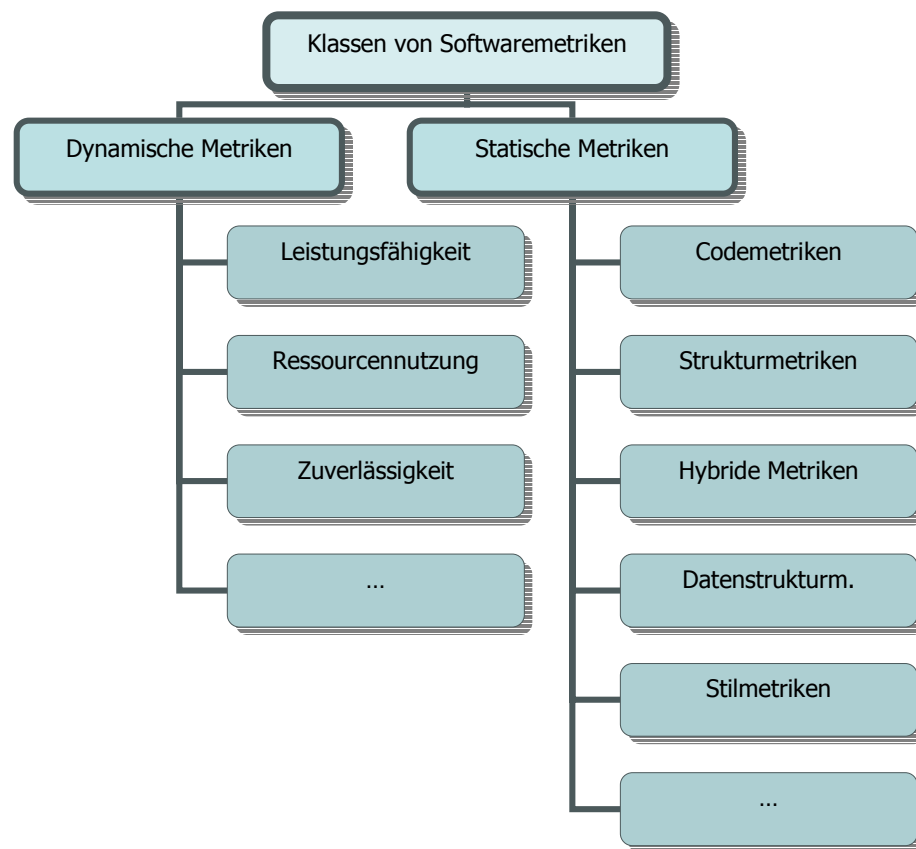


Abbildung 7 – Aspekte der Softwaremetriken

[In Anlehnung an Balzert 1998, S. 479]

3.2.5 Metriken nach Halstead

Wie bereits im vorigen Abschnitt eingeleitet, gehen die linguistischen Metriken nach Halstead auf Mitte der 70er Jahre zurück. Heute gehören sie mit den Metriken von McCabe zu den beliebtesten Maßen der Softwaremetrik und finden Anwendung in den Softwarelaboren weltweit operierender Konzerne. Die Ermittlung der Halstead – Metriken basiert auf die statische Analyse der, einem Computerprogramm zugrunde liegenden Quelltexte. Halstead definiert die Basisgrößen Anzahl der Operatoren und Operanden, die unterschiedliche Anzahl von Operatoren und Operanden, sowie die Anzahl der Programm- und Kommentarzeilen eines Quellcodes und leitet alle weiteren Kennzahlen nach diesen ab. Operatoren bilden in diesem Zusammenhang alle Codestücke eines Quelltextes, die eine Aktion an ihre

Ausführung koppeln. Beispiele hierfür sind Prozessorbefehle, arithmetische und logische Operationen, wie die Addition zweier Zahlen oder die Negierung eines Registerwertes. Operanden bilden nach Halstead's Definition alle Codeteile die Daten repräsentieren. Variablen, Konstanten oder direkte Zahlenwerte würden daher als Operanden gewertet. Auf diesen Werten aufsetzend definierte er Metriken für die Frequenz von Kommentaren, Programmlänge, Umfang des verwendeten Vokabulars, Programmvolumen, den mentalen Aufwand bei der Softwareentwicklung und Koeffizienten über die die durchschnittliche Fehlerwahrscheinlichkeit in einem Quelltext berechnet wird. Ein bedeutender Vorteil der Halstead – Messungen ist, dass einfach und effizient zu ermittelnde Größen verwendet werden die für jegliche Programmiersprachen anwendbar sind. Nachteil ist der alleinige Fokus auf die Implementierung und der, je nach Umsetzung der Definition, offene Spielraum für eigene Einstufungen der Basisgrößen Operatoren, Operanden, Kommentar- und Programmzeilen. [Balzert 1998, S. 480f; Dumke 1992, S. 91; Krell 2003, S. 20-24; Zuse 1999, S. 5]

3.2.6 Metriken nach McCabe

Weg von den reinen Implementierungsaspekten, die durch die Spracheigenschaften wesentlich beeinflusst werden können, legen die Metriken nach McCabe Hauptaugenmerk auf die logische Programmstruktur. Wesentliches Maß für McCabe ist die Einfachheit einer Programmstruktur, da er der Auffassung war, eine Software gemäß einer definierten Spezifikation erstellen zu können, wenn die Einfachheit eines Systems sichergestellt werden könne. Aufsetzend auf dem graphentheoretischen Ansatz, bedient sich seine These des Modells der gerichteten Graphen, und stellt ein Computerprogramm als eine abzuarbeitende Sequenz von Instruktionen dar. Dabei unterscheidet er nach Codeelementen, die die verschiedenen Modellelemente repräsentieren. Die Knoten eines Graphen entsprechen den abzuarbeitenden Befehlsfolgen. Die Kantenzüge, welche die Knoten wiederum verknüpfen, werden durch den möglichen Programmfluss geliefert und komplettieren so das Modell. Hinsichtlich des Programmflusses unterscheidet er zwischen linearen Codestücken, die lediglich eine sequenzielle Verarbeitung ausmachen und Sprüngen, die ein Überspringen oder ein wiederholtes Ausführen von Programminstruktionen hervorrufen können. Sprünge können wiederum in bedingte, denen eine Bedingung zugrunde liegt, und unbedingte, welche in jedem Fall ablaufen, klassifiziert werden. Ein Startknoten dient als Eingang in einen Graphen, bei dem die

Ausführung des Gesamtsystems, beziehungsweise aufgebrochen auf eine Komponente, die Ausführung des Teilprodukts gestartet wird. Mit Abarbeitung der logischen Struktur endet die Verarbeitung in einem Endknoten der auch als Programmaussprung bezeichnet wird. Aufbauend auf das Modell entwickelte McCabe Kennzahlen, die die Eigenschaften des Programms beschreiben. Beispiele hierfür sind die Anzahl der Knoten und Kanten, die Anzahl der so genannten „toten“ Knoten und Kanten, die in einem Programmfluss niemals durchlaufen werden, und die Anzahl der Ausgangsknoten aus einem Programm. Dies alles sind Metriken, die die logische Struktur beschreiben und Indikatoren für die Programmkomplexität sein können. Die bekannteste Metrik von McCabe ist die zyklomatische Zahl mit:
 $e := \text{Anzahl der Kanten}$ und $n := \text{Anzahl der Knoten}$

$$V(G) = e - n + 2$$

Abbildung 8 – McCabe's zyklomatische Komplexität

Der Summand „2“, im letzten Teil der Berechnung, ist ein fester Wert für die Anzahl der Ein- und Aussprünge im betrachteten Programmteil. McCabe geht dabei von einer in seinen Augen „guten“ Programmstruktur, mit einem Einsprung und einem Aussprung, aus. Vorteil der Messung ist die Objektivität bezüglich des untersuchten Teilprodukts durch das vorig beschriebene Modell der gerichteten Graphen, beziehungsweise Kontrollflussgraphen. Aus diesem sind außerdem Testfälle für alle Kantenzüge herzuleiten, worüber Vollständigkeit beim Aspekt des Testens gewährleistet werden kann. Darüber hinaus lassen sich die Kennzahlen, ebenso wie die der Halstead – Metriken, leicht und effizient berechnen und führen schnell zu Ergebnissen. Nachteilig ist, dass das Programm als Untersuchungsobjekt einen zu wesentlichen Faktor einnimmt und andere Programmmerkmale, wie beispielsweise komplex verschachtelte lineare Instruktionen, durch die Zusammenfassung zu einem linear abzuarbeitenden Knoten zu wenig berücksichtigt werden. Die Gesamtkomplexität eines Systems kann damit nur teilweise dargestellt werden und lässt daher lediglich bedingte Aussagen über die Eigenschaften eines Gesamtsystems zu. [Balzert 1998, S. 481f; Dumke 1992, S. 102f; Fenton, Pfleeger 1997, S. 293f; Halstead 1977, Part I; Krell 2003, S. 24-28; Zuse 1999, S. 5]

3.2.7 Metriken objektorientierter Programmierparadigmen

Besonders am Beispiel der Halstead – Metriken lässt sich verdeutlichen in wie weit eine verwendete Programmiersprache die Messwerte von Softwaremetriken beeinflusst. Produkte, denen verschiedene Technologien zugrunde liegen, lassen sich dementsprechend nicht ohne weitere Betrachtungen vergleichen. Besonders wenn der Technologieunterschied in verschieden verwendeten Programmierparadigmen liegt, bedürfen Metriken Anpassungen, um Falscheinschätzungen bezüglich des untersuchten Codes zu verhindern. Softwaremessungen für objektorientierte Programmiersprachen wie Java, Delphi und C++ bedürfen darüber hinaus Analysen über Klassenstrukturen, Objekten, Schnittstellen und Klassenimplementierungen. Ergänzende Metriken können Klassenhierarchie- und Informationsmaße, Strukturiertheits-, Lesbarkeits- und Erweiterbarkeitsmaße sein, die besonders die Aspekte der Instanziierung, Polymorphie, Vererbung und Wissensverteilung betreffen. [Dumke 1992, S. 137f, 141-143]

3.3 Anwendung, Ziele und Grenzen der Softwaremessung

Damit Metriken zu einem praxistauglichen Instrument der Qualitätssicherung werden können, müssen die ermittelten Ergebnisse eine Reihe von Kriterien erfüllen. Wie bei jeder in einer Wissenschaft verwendeten Messung ist es notwendig, dass das Maß objektiv ermittelt wurde und frei von subjektiven Einflüssen des Durchführenden ist. Darüber hinaus muss ein Maß zuverlässig sein, was meint, dass die Wiederholung des Vorgangs unter den entsprechend gleichen Bedingungen wiederum die zuvor gewonnenen Werte erzielt. Außerdem müssen die Werte dem Grundsatz der Validität genügen, was einen direkten Rückschluss auf die Kenngrößen ermöglicht. Bezüglich der Definition der Messung müssen die Ergebnisse auf einer Skala aufgetragen werden können um sie mit anderen Werten vergleichbar darstellen zu können. Ökonomie und Nützlichkeit bilden die wirtschaftlichen Faktoren in Bezug zu Softwaremetriken. Dabei beschreibt die Ökonomie wie leicht und kostengünstig Metriken berechnet werden können. Diese Eigenschaft ist maßgeblich vom Automatisierungsgrad der Erstellung abhängig. Die Nützlichkeit ist das Attribut, welches die Erstellung und die Verwendung der Kennzahlen gestattet. Werden Kennzahlen als nicht verwendbar und unnütz eingestuft werden sie im Qualitätssicherungsprozess nicht verwendet. Nach Dumke [Dumke 1992, S. 40] werden die Eigenschaften Objektivität, Zuverlässigkeit und Validität als Hauptgütekriterien eingestuft, hingegen

die weiteren Kennzahlen den Charakter von Nebengütekriterien haben. [Balzert 1998, S. 227]

Werden Softwaremetriken nach diesen Gütekriterien eingestuft und für ein Produkt relevante ausgewählt, ist das Ziel ein wirtschaftliches Werkzeug zu finden, mit Hilfe dessen die Kennzahlen ökonomisch ermittelt werden können. Geeignet eingesetzt, verhelfen Metriken Softwareingenieuren einen schnellen Einstieg in komplexe Systeme zu gewinnen. Da die Kennzahlen vom eigentlichen Quelltext abstrahieren, erhalten Gutachter über ein abstraktes Modell der zugrunde liegenden Software, einen Überblick über Kernkomponenten und komplexe Codesequenzen dieser. Sie geben konstruktive Anhaltspunkte, auf denen Reengineeringprozesse ansetzen können, um die Qualität des Gesamtprodukts zu verbessern. Nach Änderungen können sie verwendet werden, um die vorher gefundenen Mängel auf ihre Behebung zu prüfen. Bei Neuentwicklungen stellen Softwaremessungen Richtlinien für die Implementierung dar, indem ihre Soll- und Grenzwerte den spezifizierten Softwareanforderung angepasst werden, so dass bei einer Prüfung Defizite übersichtlich präsentiert werden. So gesehen dienen sie in dieser Funktion als so genannte „style“ oder „design – guides“ dem Systementwurf und der Umsetzung. Sie verhelfen den Ist – Status eines Projekts zu erfassen und können somit als frühzeitige Indikatoren für das Qualitäts- und Projektmanagement dienen. Softwaremetriken können als Ausgangspunkte und Diskussionsgrundlagen in Prüfungen und Abnahmen zwischen Entwicklungsphasen verwendet werden. Angewandt über einen vollständigen Produktlebenszyklus lassen sich Trends und Fortschritte der einzelnen Produkte, beziehungsweise zwischen einzelnen Releases untersuchen und analysieren, und somit wiederum als Anstoß für Änderungen und Anpassungen verwenden. [Fenton, Pfleeger 1997, S. 12f; Gilb 1977, S. 23; Lewerentz et al 2000, S.66f]

Neben den Vorteilen und praktischen Anwendungsszenarien von Softwaremessungen lassen sich ebenso gravierende Probleme mit solchen Kennzahlen festhalten. Häufig bereitet die korrekte Interpretation der erzielten Messwerte Probleme, so dass die Gefahr besteht, die Indikatoren falsch zu werten und entsprechend falsche Maßnahmen einzuleiten. Ist dies der Fall wird die Bedeutung und eigentliche Repräsentation der Kennzahlen nicht korrekt und vollständig herausgelesen und führt zu Verwirrung. Eine sorgfältige Analyse der Messwerte ist daher zwingend erforderlich, um nicht falsche Rückschlüsse daraus zu ziehen und fehlgeleitet neue Aufgaben im Entwicklungsprozess zu spezifizieren. [Dumke 1992, S. 33f; Orci 2001, S. 78; Sommerville 2003, S. 563f]

4 Softwareprüfung und Anforderungen an dabei verwendete Softwarewerkzeuge

4.1 Zertifizierung sicherheitsrelevanter Software

Bisher wurden Softwaremetriken wesentlich im Zusammenhang zum hausinternen Entwicklungsprozess und dessen Qualitätsgewährleistung betrachtet. Besonders bei sicherheitsrelevanter Software spielen externe Prüfungen eine maßgebliche Rolle. Zur Gewährleistung der spezifizierten Gütekriterien werden Softwareprodukte von externen unabhängigen Gutachtern geprüft, womit die Ergebnisse der Prüf- und Zertifizierungsstellen aktiv in den Entwicklungsprozess eingehen. Das Berufsgenossenschaftliche Institut für Arbeitsschutz (BIA) bildet eine solche Prüf- und Zertifizierungsstelle und arbeitet diesbezüglich mit entwickelnden Unternehmen zusammen.

4.1.1 Das Berufsgenossenschaftliche Institut für Arbeitsschutz

Als Abteilung des Hauptverbands der Berufsgenossenschaften (HVBG) ist das BIA ein Institut, welches vorrangig den gewerblichen Berufsgenossenschaften und dazugehörigen Instituten in naturwissenschaftlichen und technischen Fragen zu Arbeits- und Gesundheitsschutz zur Seite steht. Hauptaufgaben kommen aus den Bereichen der Forschung, Entwicklung, Untersuchung und Zertifizierung von Produkten und Stoffen, Beratungsfunktionen und Mitwirkung bei Normungen. Im Zusammenhang mit der Zertifizierung von Software ist der Fachbereich 5: „Unfallverhütung – Produktsicherheit“ zuständig. Die Referate „Neue Technologien, Mensch und Technik“ und „Schutz- und Steuereinrichtungen“ arbeiten diesbezüglich mit gewerblichen Betrieben in der Produktentwicklung zusammen. Häufig zu findende Beispielanwendungen sind Steuerungen an Maschinen und Anlagen, sowie Sicherheitseinrichtungen wie Lichtschranken und -gitter zum Personenschutz. [HVBG 2004]

4.1.2 Der Zertifizierungsprozess im BIA

Um Fehlern im Entwicklungsprozess und Gesamtprodukt vorzubeugen, wird häufig die Zusammenarbeit mit Prüf- und Zertifizierungsstellen als beratende Instanzen und externen Gutachtern gesucht. Dabei steht die offene Kooperation und Kommunikation im Vordergrund, da sie die normkonforme Produktentwicklung und –qualität ermöglicht. Auf diesem Weg sichert sich der Hersteller ab, keine weitrei-

chenden Änderungen zu einem fortgeschrittenen Zeitpunkt am Produkt umsetzen zu müssen und ermöglicht der prüfenden Instanz eine entwicklungsbegleitende Prüfung über den gesamten Produktlebenszyklus hinweg. Bei der Kooperation arbeiten beide Instanzen von der ersten Phase der Entwicklung an gemeinsam am Produkt. Teilprodukte der einzelnen Phasen werden dabei vom Hersteller an die prüfende Organisation gegeben, um diese abnehmen zu können. Am Beispiel heißt das, dass frühzeitig Lastenhefte des Produkts von Prüfern der zertifizierenden Organisation analysiert werden, bevor der Hersteller darauf aufsetzend Pflichtenheft und Produktdesign entwickelt. Werden die Teilprodukte von der Prüfstelle zertifiziert, bilden sie die Grundlage für weitere Entwicklungen. Bei den Prüfungen durch Ausschüsse, die sich aus Gutachtern einer Zertifizierungsinstanz zusammensetzen, wird vornehmlich die Übereinstimmung der Folgeprodukte mit der anfänglichen Spezifikation und die konsequente Umsetzung der Sicherheitsauflagen, die zur Zertifizierung des Produkts nötig sind, überprüft. Ist der Entwurf konform den spezifizierten Anforderungen wird seitens des Herstellers der Prototyp entwickelt, welcher zum Testobjekt der Zertifizierungsstelle wird. An ihm werden systematisch zuvor spezifizierte Tests durchgeführt. Sind diese erfolgreich, werden darauf aufsetzend auf einem Serienmodell Umgebungstests und im letzten Schritt Akzeptanztests vollzogen. Gemäß den Prüfungen werden Prüfberichte erstellt, die die Ergebnisse dokumentieren und bei einer positiven Prüfung zur Zertifizierung des Produkts führen.

Handelt es sich bei dem Prüfling um ein Softwareprodukt, wird seitens des BIA und ähnlichen Institutionen ein statisches Analysewerkzeug (Abbildung 7) bei der Prüfung des Prototypen verwendet. Die Resultate der Softwaremessung dienen den Gutachtern beim Einstieg und der Analyse des Prüflings. Darauf aufsetzend werden sie zur Kommunikationsgrundlage für Prüfer und Entwickler. Finden innerhalb des Produktlebenszyklus Änderungen am Produkt selbst statt, müssen bereits durchgeführte Phasen der Zertifizierung wiederholt werden, um zu prüfen, ob die Änderungen seiteneffektfrei im bisher entwickelten Produkt arbeiten und das Gesamtsystem den Sicherheitsanforderungen noch genügt. [Dumke 1992, S. 9; Krell 2003, S. 7f; Reinert, Reuß 1991, S. 1, 3ff]

4.2 Hilfsmittel beim Zertifizierungsprozess – Die Anwendung LOGISCOPE

Gilt es im Zertifizierungsprozess Software zu untersuchen, erhalten die Prüfer unter Anwendung geeigneter Rechnerunterstützung einen schnellen Einstieg und einen

grogen Überblick über den Prüfling. Sicherheitsrelevante Software bedarf oft einiger tausend Zeilen Quellcode zur Implementierung der Sicherheitsfunktionen, was eine Unterstützung durch Rechner zwangsläufig nötig macht, um eine effiziente Prüfung vollziehen zu können. Zu diesem Zweck wird im BIA die Anwendung LOGISCOPE der Firma VERILOG eingesetzt. Sie ist ein Analyse- und Bewertungswerkzeug, das Software bezüglich seiner Komplexität im Aufbau und herunter gebrochen auf Instruktionsniveau untersucht. Wesentliche Einsatzgebiete von LOGISCOPE sind die Softwareentwicklungen für die Einsatzbereiche der Telekommunikations-, Luft- und Raumfahrt- und Kernkraftindustrie. Eine aktuelle Weiterentwicklung der Applikation LOGISCOPE wird in Form der Anwendung Telelogic TAU / Logiscope auf dem Markt für Unix – und Microsoft Windows – Plattformen angeboten.

In der im BIA eingesetzten LOGISCOPE – Version werden die Softwaremessungen nach den Halstead und McCabe – Metriken umgesetzt und können anhand verschiedener graphischer Darstellungen visualisiert werden. Die Repräsentation von Programmstrukturen wird über Kontrollflussgraphen (KFG, Kapitel 3.2.4, Abbildung 9), die auf die Metriken von McCabe aufsetzen, ermöglicht. Weitere Darstellungsformen bilden die Kiviatgraphen und die Call – Graphen.

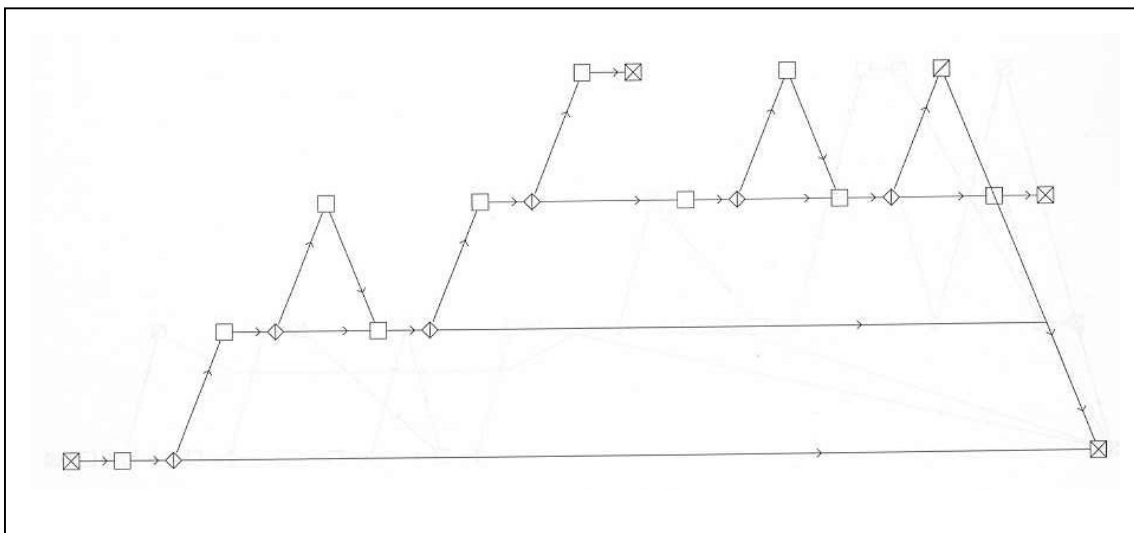


Abbildung 9 – Beispiel eines Kontrollflussgraphen in LOGISCOPE

Erstere stellen die Einzelmetriken (Abbildung 10) und darauf aufsetzend die Qualitätskriterien auf Komponentenebene anhand vier Quadranten dar (Abbildung 11). Wesentliche Bestandteile des Kiviatgraphen sind der äußere und innere Kreis, die jeweils die Ober- und Untergrenze der Einzelmetriken bezüglich des Qualitätsmo

dells widerspiegeln. Tritt eine Grenzüber- oder –unterschreitung auf, ist diese im Kivatmodell gut ersichtlich.

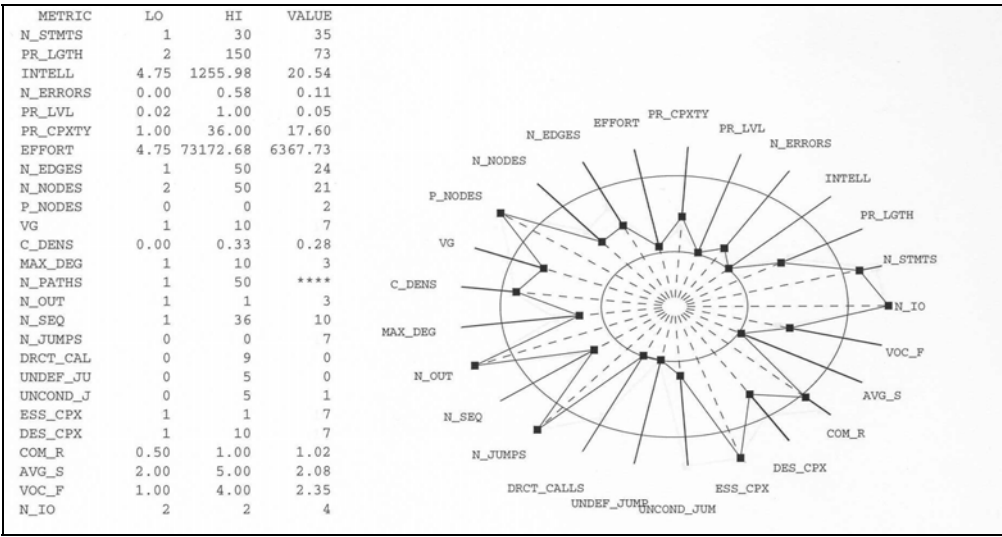


Abbildung 10 – Beispiel eines Kiviattgraphen in LOGISCOPE

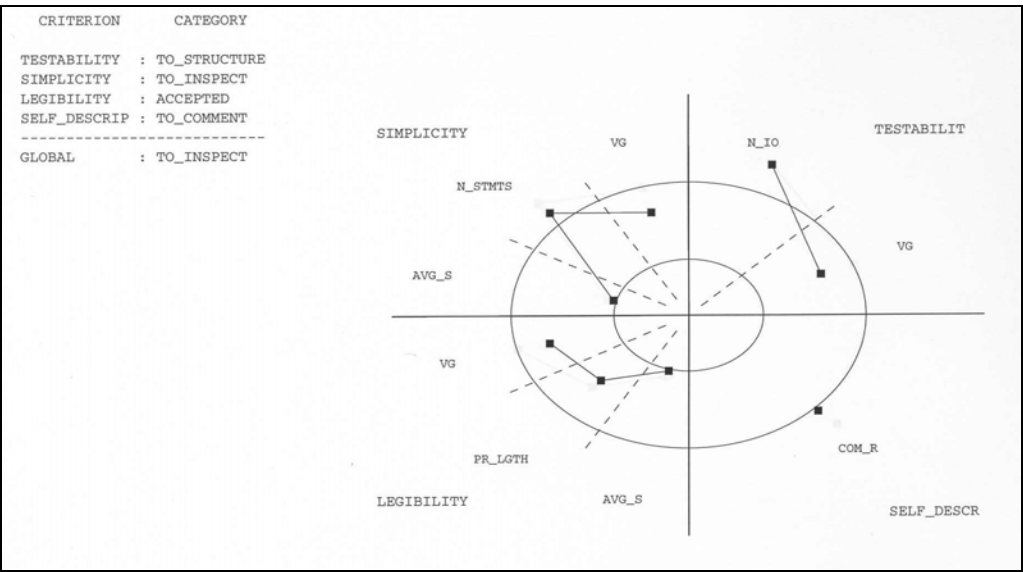


Abbildung 11 – Beispiel eines Qualitätsmodells in LOGISCOPE

Die Call – Graphen (Abbildung 12) repräsentieren die Aufrufstruktur der Teilprodukte untereinander und somit die Aufrufe des Gesamtprodukts.

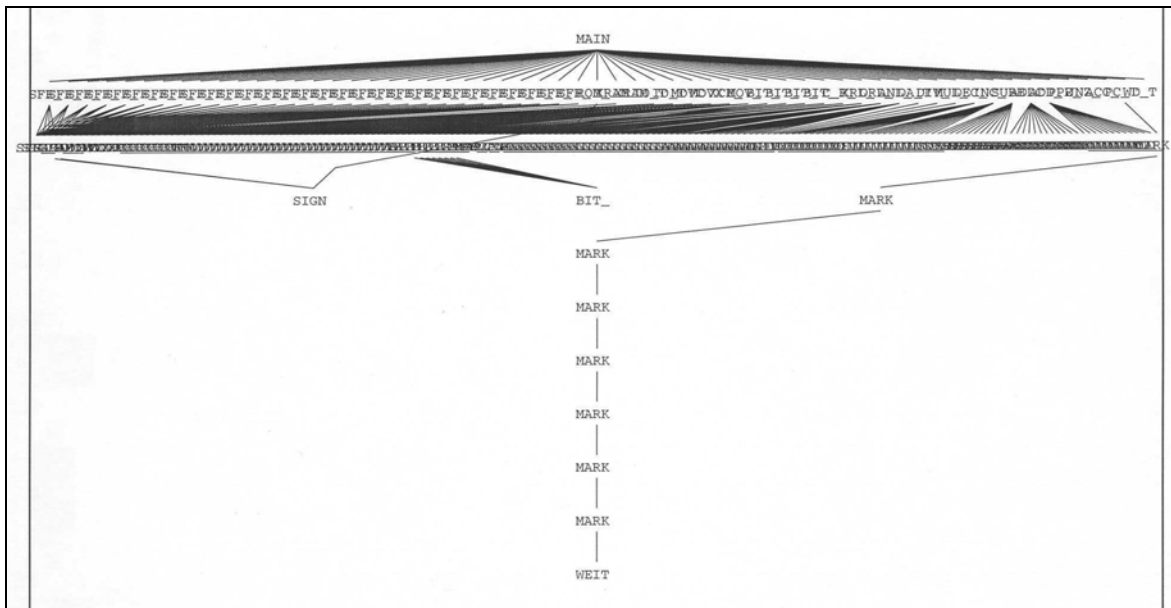


Abbildung 12 – Beispiel eines Aufrufgraphen in LOGISCOPE

Die nach der Norm DIN ISO 9126 spezifizierten und in Kapitel 3.2.3 beschriebenen Qualitätskriterien werden in LOGISCOPE durch vier definierte Kriterien ansatzweise abgebildet. Sie sind als „Testbarkeit“, „Einfachheit“, „Lesbarkeit“ und „Selbstbeschreibung“ angelegt und werden aus gewichteten Einzelmetriken, wie zum Beispiel der zyklomatischen Zahl, der Anzahl der Modulein- und –aussprünge und der Anzahl der Instruktionen von LOGISCOPE berechnet. Über die Gewichtungen können prozentuale Werte für die Kriterien erzielt werden, die wiederum zu Aussagen führen, ob das Kriterium akzeptiert wurde oder Nachbesserungen über beispielsweise Dokumentationen oder Inspektionen zu leisten sind. Die visuelle Darstellung der Qualitätskriterien erfolgt über spezielle Graphen, ähnlich der Kiviatgraphen, die einen Quadranten für jedes Kriterium aufweisen. Die Umsetzung der Qualitätskriterien ist eine Besonderheit der LOGISCOPE – Applikation, die nicht viele Softwareprodukte in diesem Anwendungskontext teilen.

Wie auch im BIA, wird das Analysewerkzeug vornehmlich zum Einstieg in Reviews und zur Bewertung der Softwarequalität von Herstellern, externen Gutachtern und Kunden verwendet. [Dumke 1992, S. 193f; Krell 2003, S. 11-19, 28 – 31; Reinert, Reuß 1991, S. 1; VERILOG 1993b, S. 4.3 – 4.33; Windpassinger 2000, S. 135 – 142, 146]

4.3 Evaluierung und Weiterentwicklung der eingesetzten Qualitätssicherungsinstrumente von Krell

Im Rahmen der Diplomarbeit „Bestimmung von Qualitätskriterien für sicherheitsrelevante Software im Maschinenschutz auf Basis von zertifizierten Industrieanwendungen“ von Michael Krell [Krell 2003] wurden in Kooperation mit dem Berufsgenossenschaftlichen Institut für Arbeitsschutz die Umsetzung der Qualitätskonzepte durch die Software LOGISCOPE analysiert, bewertet und darauf aufsetzend die Modelle weiterentwickelt. In einem ersten Schritt galt es dabei die implementierten Metrikkonzepte von Halstead und McCabe zu untersuchen und deren Realisierung in LOGISCOPE zu prüfen. Als Untersuchungsgegenstand wurden dabei Assemblercodes gewählt. Sie erschienen als geeignete Prüfgrundlage, da sie nach wie vor sehr häufig zur Programmierung von Sicherheitseinrichtungen an Industrieanlagen verwendet werden, und entsprechend oft Gegenstand von Zertifizierungen des BIA sind. Die Analyseergebnisse von LOGISCOPE wurden dabei mit den Beschreibungen der Metriken aus den Handbüchern geprüft und deren Aussagekraft festgestellt und bewertet. Ausgehend von den Metriken wurden die Qualitätskriterien als Faktoren des Qualitätsmodells aus Kapitel 3.2.3 und die daraus wiederum resultierende globale Qualität des Gesamtprodukts untersucht und bewertet.

Die dabei entstandenen Resultate erörtern Schwachstellen der Metriken selbst und in deren Implementierung, so dass sie als Grundlage für Weiterentwicklungen der Konzepte dienen. Um einen direkten Vergleich zwischen den in LOGISCOPE implementierten und von Krell weiterentwickelten Metriken darstellen zu können, wurde ein Werkzeug „EXPORT“, basierend auf Visual Basic und Microsoft Excel 2000 entwickelt. Es ermöglicht die Ergebnisse der LOGISCOPE – Analysen in ein Exceldokument einzulesen und die von Krell weiterentwickelten Kennzahlen, aufsetzend auf den Basismaßen von Halstead und McCabe zu berechnen und den LOGISCOPE – Ergebnissen gegenüberzustellen. Durch diesen Prozess und den direkten Vergleich der verschiedenen Interpretationen der Kennzahlen wurden eine Reihe von Änderungsvorschlägen gegenüber dem bisher verwendeten Analysewerkzeug erarbeitet. Teils ermöglicht LOGISCOPE die Änderungen umzusetzen, wie beispielsweise Anpassungen der Ober- und Untergrenzen, die für die Bestimmung der Qualitätskriterien gemäß der Norm DIN ISO 9126 benötigt werden. Jedoch konnten weiterführende Anpassungen, welche Änderungen im Analysatormodul bedingen, nicht umgesetzt werden.

4.4 Anforderungen seitens des BIA an eine zukünftige Analysesoftware gegenüber dem bisher eingesetzten Werkzeug

Um Anforderungen und umzusetzende Konzepte einer zukünftigen Rechnerunterstützung von Zertifizierungsprozessen zu erläutern, wird im ersten Schritt die Architektur von LOGISCOPE dargestellt. Danach wird der aktuelle Ist – Zustand einer Softwareuntersuchung visualisiert und an einem Beispielprozess verdeutlicht, um darauf basierend Defizite in den Einzelprozessen darzustellen. Zur graphischen Illustration des Geschäftsprozesses wird konzeptionell an die ARIS – Methode angelehnt, die häufig in der Wirtschaftsinformatik zur Geschäftsprozessmodellierung und –optimierung anhand Ereignisgesteuerter Prozessketten (EPK) eingesetzt wird. Sie integriert die verschiedenen Aspekte der Funktionen, Daten, Organisationseinheiten und Leistungen eines Geschäftsprozesses und verknüpft somit die verschiedenen Sichten auf den Gesamtprozess. In den folgenden Abbildungen wird das ARIS – Konzept durch Symbole der „Unified Modelling Language“ (UML) erweitert. Für gewöhnlich werden diese Zeichen im Kontext von UML – Sequenzdiagrammen zur Repräsentation von Objekten und deren Lebensdauer verwendet. UML ist eine moderne Methode Softwarestrukturen graphisch zu visualisieren, spezifizieren, konstruieren und zu dokumentieren.

Die folgende Abbildung zeigt die bei der Darstellung des Ist – und des Soll – Zustands verwendeten Symbole.

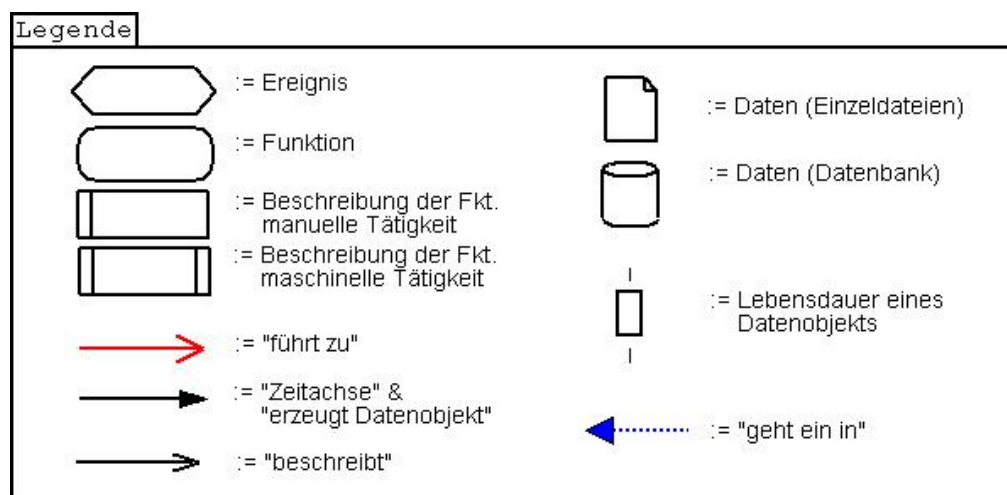


Abbildung 13 – Verwendete Symbole zur graphischen Visualisierung

Die Erläuterungen zum Ablauf des Prozesses werden exemplarisch an einem zu untersuchenden Assemblercode aufgezeigt.

4.4.1 Die Architektur der Analysesoftware LOGISCOPE

Die LOGISCOPE – Architektur sieht vor in einem ersten Schritt die zu untersuchenden Quelldateien in das Analysemodul einzugeben. Dort werden sie geprüft, woraus die Ergebnisdateien und die Zähldateien erzeugt werden. Erstere enthalten einige wenige Untersuchungsergebnisse, die benötigt werden um die Komplexitätsmessungen und die graphischen Darstellungen zu berechnen. Dieses wird vom statischen Editor geleistet, welcher auf die zuvor generierten Ergebnisdateien als Grundlage zugreift. Ist eine Archivierung der Ergebnisdateien gewünscht, kann diese durch das Archivwerkzeug geleistet werden. Eine exemplarische Ergebnisdatei wird in Abbildung 17 dargestellt, da im folgenden Kapitel weitere Eigenschaften zum bisherigen Prozess einer Softwareprüfung beleuchtet werden.

Die folgende Abbildung zeigt den schematischen Aufbau und die Funktionsweise des LOGISCOPE – Analysewerkzeugs:

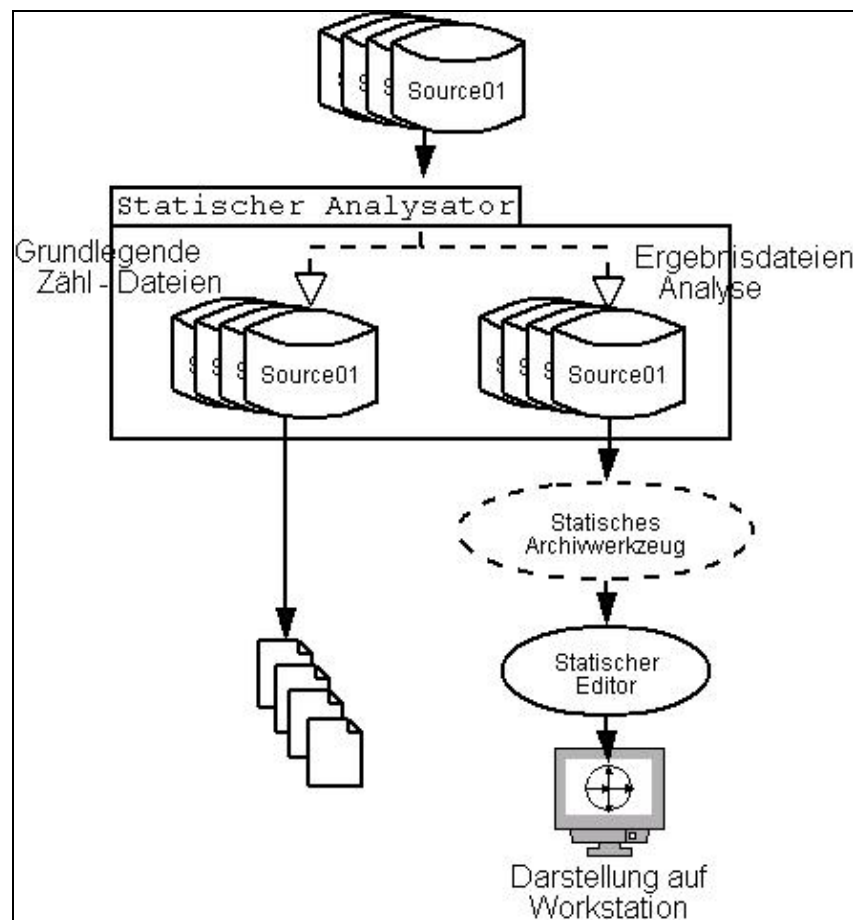


Abbildung 14 – Architektur LOGISCOPE

[In Anlehnung an VERILOG 1993a, S. 1.7]

Alle weiteren Basiskennzahlen für die Halstead – und McCabe – Metriken sind in den Zähldateien enthalten. Dort liegen sie für jede Komponente in tabellarischer Form vor. Somit bilden sie die Grundlage aller aufsetzenden Berechnungen von Kennzahlen.

```
+ Reference File for ASM68XX_INTER Assembler
+ Coding of the "type of LOGISCOPE action to be performed" field can
+ have the following values :
+
+ 0 : instruction ignored ;
+ 1 : instruction analyzed ;
+   no particular action ;
+ 2 : procedure call :
+   a parameter must specify a module name ;
+ 3 : module exit other than normal end ;
+ 4 : conditionnal jump :
+   one or several parameters must specify the branch point ;
+ 5 : unconditionnal jump :
+   one or several parameters must specify the branch point ;
+ 6 : while loop ;
+ 7 : end while loop ;
+ 8 : while loop exit :
+   possibly with the loop label as parameter ;
+ 9 : while loop reset :
+   possibly with the loop label as parameter :
+
+ ...
+
+ I + BCC      : 4 1
+ I + BCLR    : 1
+ I + BCS     : 4 1
+ I + BEQ     : 4 1
+ I + BGE     : 4 1
+ I + BGT     : 4 1
+ I + BHCC    : 4 1
+ I + BHCS    : 4 1
+ I + BHI     : 4 1
+ I + BHS     : 4 1
+ I + BIH     : 4 1
+ I + BIL     : 4 1
+ I + BIT     : 1
+ I + BITA    : 1
+ I + BITB    : 1
+ I + BLE     : 4 1
+ I + BLO     : 4 1
+ I + BLS     : 4 1
+ I + BLT     : 4 1
+ I + BMC     : 4 1
+ I + BMI     : 4 1
+ I + BMS     : 4 1
+ I + BNE     : 4 1
+ I + BPT     : 4 1
+
+ ...
```

Abbildung 15 – Auszug einer Referenzkonfiguration in LOGISCOPE

Um die Kennzahlen auf Basis der Quelltexte aufstellen zu können, müssen, neben den Quellen, Konfigurationen mitgeliefert werden, die beschreiben, wie die Quellen zu interpretieren sind. LOGISCOPE implementiert diese Funktion über die so genannte Referenzkonfiguration. Dabei muss der Anwender die in einem Quelltext vorkommenden und für die Berechnungen wesentlichen Ausdrücke anhand einer

Referenzdatei klassifizieren (Abbildung 15). Bei Fertigstellung bilden diese mit den zu analysierenden Quelltexten die Grundlage der Untersuchungen. [VERILOG 1993a, S. 1.7ff, 4.3 – 4.8]

4.4.2 Ein exemplarischer Prozess einer Softwareprüfung unter Verwendung von LOGISCOPE

Da über das Analysewerkzeug Quelltexte verschiedener Programmiersprachen und Softwareparadigmen geprüft werden, startet die Softwareprüfung mit der manuellen Analyse des Quellcodes durch den Gutachter, um die Spracheigenschaften im Analysemodul anhand der Referenzkonfiguration einzustellen. Betrachtet man als Prüfling einen Assemblercode, wird deutlich, dass entsprechende Änderungen der so genannten Referenzdaten sehr häufig vollzogen werden müssen. Dabei werden in LOGISCOPE die einzelnen Schlüsselwörter in beispielsweise „Instruktionen zu analysieren“, „Instruktionen zu ignorieren“, „Prozeduraufruf“, „Modulende“, „bedingter Sprung“ und „nicht bedingter Sprung“ klassifiziert. Wie im vorigen Absatz aufgezeigt, sind die Angaben für das Analysemodul wesentlich, um die untersuchten Codestücke zu gruppieren und daraufhin die Metriken erstellen zu können.

Ferner erkennen wir aus Abbildung 16, dass dieser Teilprozess der Referenzkonfiguration bei LOGISCOPE vollständig manuell abläuft und der Gutachter keinerlei Rechnerunterstützung seitens LOGISCOPE dabei erhält. Aufgabe des Prüfers ist in diesem Schritt, die Schlüsselwörter der verwendeten Programmiersprache in die Referenzdatei von LOGISCOPE zu schreiben und Klassifizierungen, gemäß Abbildung 15, zuzuordnen. Um die Vollständigkeit der Eintragungen zu überprüfen, wird iterativ immer wieder ausprobiert, ob LOGISCOPE bei der Analyse der Quelltexte Warnungen ausgibt. Wird eine Warnung generiert, heißt dies für den Prüfer, dass er Änderungen in der Referenzdatei vornehmen muss. Häufig nimmt dieser Prozess, aufgrund seiner Fehleranfälligkeit und der manuellen Durchführung mehrere Stunden, in manchen Fällen sogar mehrere Arbeitstage in Anspruch.

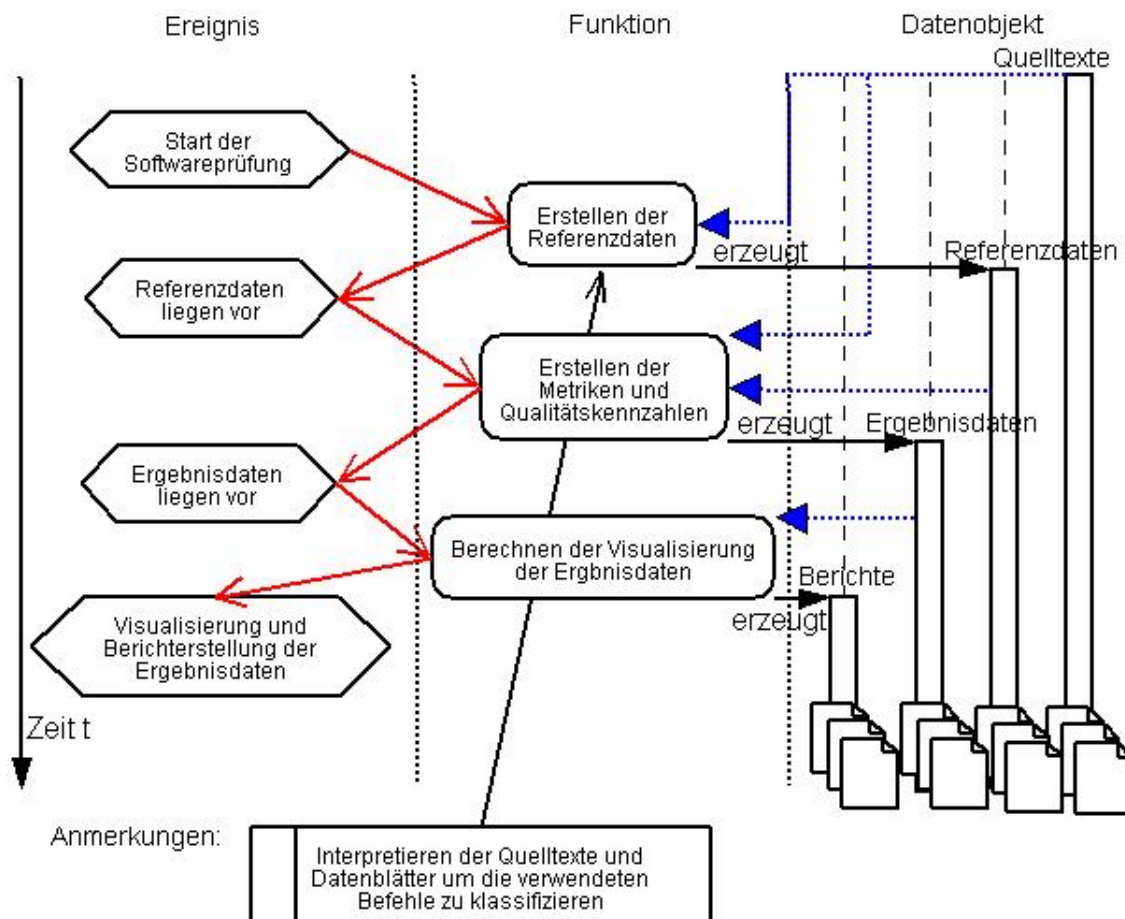


Abbildung 16 – Einbettung des Analysewerkzeugs LOGISCOPE in eine Softwareprüfung des Berufsgenossenschaftlichen Instituts für Arbeitsschutz

Liegen die Referenzdaten in Form der Referenzdatei vor und kann die Analyse ohne Warnungen durchgeführt werden, sind Quelltexte und Referenzdatei zur statischen Analyse fertig präpariert. Dabei greift das, je nach verwendeter Programmiersprache unterschiedliche, Analysemodul von LOGISCOPE auf diese Grundelemente zu und berechnet daraus die Metriken. Die Basiskennzahlen der Halstead – Metriken werden dabei in eine Ergebnisdatei und eine Zähldatei geschrieben, die seitens LOGISCOPE generiert werden. Als Grundlage für die McCabe – Metriken und die graphischen Darstellungen der Programmstruktur, fließt zusätzlich zu den Kennzahlen eine Art Pseudocode vom untersuchten Quelltext in die Ergebnisdaten ein. Dieser, vom Problem abstrahierte Pseudocode, beschreibt den Aufbau des analysierten Programms und dient als Fundament für die Visualisierung durch einen Kontrollflussgraphen. [Krell 2003]

```
1 <archive V2.0>
2 *NA*
3 SOURCE
4 .EF.
5 LANG/ASSEMBLEUR
6 *CV*
7 CURRENT_VERSION
8 *CM*
9 CURRENT_VERSION
10 TESTKOMPONENTE 96/03/25-07:14:50
11 *MO*
12 TESTKOMPONENTE 96/03/25-07:14:50
13 .EM.
14 LANG/ASSEMBLEUR
15 .DT
16 INST      12
17 COMM      13
18 ARCS      0
19 NOEU      0
20 OOPN      16
21 OPEN      8
22 OOPT      12
23 OPET      8
24 .DC.
25 !start:RIEN /1; #27.BeispielCodeAssembler.ASM,
26 :IFNS /start01; #28.BeispielCodeAssembler.ASM,
27 :RIEN /1; #29.BeispielCodeAssembler.ASM,
28 :SAUT /ende; #30.BeispielCodeAssembler.ASM,
29 !start01:IFNS /start02; #34.BeispielCodeAssembler.ASM,
30 :RIEN /3; #35.BeispielCodeAssembler.ASM,
31 :SAUT /ende; #38.BeispielCodeAssembler.ASM,
32 !start02:RIEN /3; #41.BeispielCodeAssembler.ASM,
33 /ende:FIN ; #46.BeispielCodeAssembler.ASM,
34 .DA.
35 *FI*
36
```

Abbildung 17 – Beispiel einer Ergebnisdatei in LOGISCOPE

Das Beispiel einer Ergebnisdatei aus Abbildung 17 zeigt die Metriken innerhalb der oberen Markierung. Der untere Teil führt den Pseudocode an, der seitens LOGISCOPE ebenfalls in die Ergebnisdatei geschrieben wird. Werden die Graphiken in der LOGISCOPE – Software vom Benutzer angefordert, müssen diese mit jedem Zugriff erneut vom statischen Editor (Abbildung 14) berechnet werden. Um die generierten Ergebnisse als Diskussionsgrundlage in Reviews zum Produkt verwenden zu können, besteht lediglich die Möglichkeit die Darstellungen mit den grundlegenden Metriken auszudrucken.

4.4.3 Schwachstellen unter Verwendung von LOGISCOPE und Anforderungen an eine zu entwickelnde Rechnerunterstützung

Durch die Entscheidung eine Eigenentwicklung bezüglich eines Analysewerkzeugs voranzutreiben ergeben sich eine Vielzahl von Vorteilen. Folgend werden die be

deutendsten exemplarisch geschildert, um nur einige Beweggründe zur Individuallösung darzustellen.

- Wie in Kapitel 4.3 aufgezeigt, fanden eine Reihe von Weiterentwicklungen bezüglich der in LOGISCOPE implementierten Metriken statt. Da LOGISCOPE nur bedingte Möglichkeiten zur Anpassung an die erarbeiteten Ergebnisse erlaubt, sind die weiteren Resultate in Algorithmen zu implementieren. Dies erfordert mindestens eine modulweise Erweiterung der LOGISCOPE – Implementierung oder gar die Umsetzung durch eine vollständige Neuentwicklung.
- Die LOGISCOPE – Software sitzt auf einer UNIX – Plattform als Workstation auf und bietet durch ihre Abgeschottetheit und fehlenden Schnittstellen keinerlei Anbindung zu gängigen Büroanwendungen. Solche könnten zum Beispiel für weiterführende Auswertungen und Analysen, ein umfangreicheres Berichtswesen, automatisierte Datensicherung und die effiziente und fortschrittliche Kommunikation genutzt werden.
- LOGISCOPE sieht in seiner Architektur keine Verwaltung von allgemeinen Projektdaten und lediglich eine dezentrale Haltung der Einstellungs- und Ergebnisdaten anhand einzelner Klartextdateien auf Verzeichnisebene vor. Außerdem erschwert die im BIA eingesetzte, nicht vernetzte, Konfiguration der Workstation die Datensicherung, so dass ein Ausfall der Workstation zum Verlust der Daten in digitaler Form führen könnte. Eine moderne Frontend – Backend – Aufteilung einer Neuentwicklung würde ermöglichen, dass das Frontend auf dem Client eines Gutachters installiert wird und die Projektdaten in zentraler Form als Backenddaten in einer Datenbankdatei auf einem Netzlaufwerk oder dem Clientrechner selbst abgelegt werden. Ferner ermöglicht eine solche Architektur leicht Datensicherungsmechanismen auf die Backenddaten zugreifen zu lassen sowie die Steigerung der Robustheit des Gesamtsystems gegenüber einem Ausfall eines einzelnen Clients, der die Zertifizierungsprozesse weiterer Clients somit nicht beeinträchtigen würde.
- Ein statisches Analysewerkzeug, so wie es im BIA eingesetzt wird, hat die Aufgabe, einen schnellen Einstieg in komplexe Systeme anhand abstrahierter Modelle zu ermöglichen. Um dieser Anforderung gerecht zu werden, benötigt die Anwendung eine effiziente Verwaltung der Referenzdaten, die Grundlage zur Berechnung der Metriken und so eine Basis der Systemun-

tersuchung bilden. Das bisher eingesetzte Werkzeug kann dieser Anforderung nicht gerecht werden, da keinerlei Rechnerunterstützung beim Erstellen und Verwalten der Referenzdaten vorgesehen ist. Eine zukünftige Entwicklung könnte durch komfortable Verwaltungsmechanismen die unterstützte Bearbeitung der grundlegenden Referenzdaten realisieren. So werden auch schnelle Anpassungen auf fremde Sprachdialekte für beispielsweise Assemblerprogramme leicht durchführbar, so dass bedienende Gutachter sich schnell den wirklichen Problemstellungen widmen können.

- Um einen schnellen Einstieg in zu untersuchende Quelltexte zu erhalten, ist eine intuitive Bedienbarkeit des Werkzeugs vorteilhaft. Die bisherige Analysesoftware setzt eine umfangreichere Einarbeitung und Auseinandersetzung mit den Mechanismen voraus und ist somit nicht ohne weiteres bedienbar. Hauptfunktionalitäten der Anwendung werden via Kommandoeingaben gesteuert, was eine Benutzung der Handbücher bei sporadischer Verwendung unerlässlich macht. Eine leichtere Benutzerführung, ähnlich anderer im Institut eingesetzter Softwareanwendungen, wie die der standardisierten Büroapplikationen, ermöglichen eine intuitive Handhabung und das schnelle Erzielen von Ergebnissen. Auf diese Weise behindern langwierige Einarbeitungsphasen nicht mehr die Verwendung des Werkzeugs.
- Visuelle Darstellungsformen sind die wichtigsten Funktionalitäten zur Einarbeitung in ein zu prüfendes Softwaresystem. Sie vermitteln einen übersichtlichen und schnellen Eindruck von den „Hot Spots“ der Anwendung. LOGISCOPE stellt diesbezüglich gute Funktionalitäten zur Verfügung und ermöglicht in Betrachtung einer Codesequenz zwischen den verschiedenen Darstellungsformen zu springen. Darüber hinaus bietet es den Wechsel zwischen der abstrahierten graphischen Darstellung zur detaillierten Betrachtung eines Codeelements im Quelltext. Darstellungsfunktionalitäten sind in einer neuen Anwendung auszubauen und mit aufsetzenden Funktionswerkzeugen zu erweitern.
- Durch die Integration der Visualisierung in Büroanwendungen ergeben sich Möglichkeiten der individuellen Berichtserstellung zu den betrachteten Codestücken und die Verteilung dieser über angebundene Kommunikationsinfrastrukturen. Die Individualität könnte sich dabei durch die weitere graphische Bearbeitung eines Reports äußern, der mit Vermerken und Hinweisen versehen, via Netzwerk den Kooperationspartnern zeitnah zugestellt

werden kann. Dies zeigt, dass durch die Verwendung von standardisierten Bürowerkzeugen zur Abbildung des Berichtswesens Medienbrüche, wie sie in der vorigen Lösung durch „ausdrucken“ erzeugt werden, verhindert werden können.

- Eine, dem BIA zur Verfügung stehende Programmierschnittstelle, ermöglicht in Zukunft alle Änderungen, Anpassungen und Wartungen an einem Analysewerkzeug selbst vorzunehmen. Da seitens des Instituts daran gedacht wird, weitere Anpassungen, nicht nur im Assemblermodul, sondern vielmehr im Kontext weiterer gängiger Programmiertechnologien in der Sicherheitstechnik, vorzunehmen, bietet eine offene Programmierschnittstelle alle Änderungs- und Erweiterungsmöglichkeiten. Wird eine schnell verständliche und im Sinne des „Rapid Application Programming“ (RAP) arbeitende Programmiertechnik verwendet, ist gewährleistet, dass sich weitreichende Ergebnisse, selbst bei kurzzeitigen Weiterentwicklungen erreichen lassen.

5 Softwaredesign der Neuentwicklung

Die in Kapitel 4.4.3 aufgezeigten Schwachstellen der bisher eingesetzten Anwendung LOGISCOPE bilden die Grundlage für das Softwaredesign einer Neuentwicklung. In den folgenden Abschnitten werden die Ansätze, diesen Mängeln entgegenzuwirken, dargestellt und anhand von Graphiken visualisiert.

5.1 Vorteile einer Eigenentwicklung

Die langjährige Erfahrung der Gutachter und die Untersuchungen, die bezüglich der in LOGISCOPE implementierten Metriken von Krell [Krell 2003] durchgeführt wurden, bilden das essentielle Fundament der Neuentwicklung. Demnach wird der Fokus auf die Umsetzung der Kennzahlen, gemäß den aus der Analyse resultierenden Definitionen, und auf weitere Aspekte der Benutzer gelegt. Die weiterentwickelten Metriken bilden neue Instrumente für die Prüfer. Darüber hinaus sind die, für die Nutzer wesentlichsten Funktionen, die graphische Darstellung zur Abstraktion von den zu untersuchenden Quellcodes, entsprechend den in LOGISCOPE eingesetzten Instrumenten anzugleichen und aufsetzend neue Modelle der Visualisierung zu entwickeln. Bisher in LOGISCOPE noch nicht eingesetzte Konzepte, wie beispielsweise die Einbindung von „Bürowerkzeugen“ zur weiterführenden Illustration von Berichten und Kommunikation zwischen kooperierenden Instanzen, sind mit den Nutzern zu entwickeln und prototypisch zu implementieren. Die Programmierschnittstelle, die durch die Entwicklung in Kooperation mit dem BIA offen gelegt wird, erlaubt zukünftig alle Möglichkeiten der Weiterentwicklung und Anpassung der Applikation nach den Wünschen und Vorstellungen der Anwender. Somit wird das „Black – Box“ – Prinzip der in der Vergangenheit eingesetzten Lösung durch die Transparenz einer selbstentworfenen Individualsoftware abgelöst.

5.2 Ableitung des Soll – Zustands und dessen Softwarearchitektur aus dem Ist – Prozess

Die Struktur einer Neuentwicklung enthält als wesentlichste Elemente die zentralen Schritte aus dem Ist – Zustandsmodell. Demnach wird der Nutzer weiterhin Spracheigenschaften der zu untersuchenden Software anhand Referenzdaten konfigurieren und diese als Grundlage mit den Quelltexten dem Analysemodul zur Verfügung stellen müssen. Daraus resultieren die Kennzahlen, die permanent gesichert werden und als Basis der graphischen Abstraktion anhand verschiedener Modelle dienen.

Folgende Graphik beschreibt die Schritte und Struktur des vorgesehenen Soll – Zu

stands und nutzt dabei die bereits in Abbildung 16 verwendeten Symbole. (Legende – Abbildung 13)

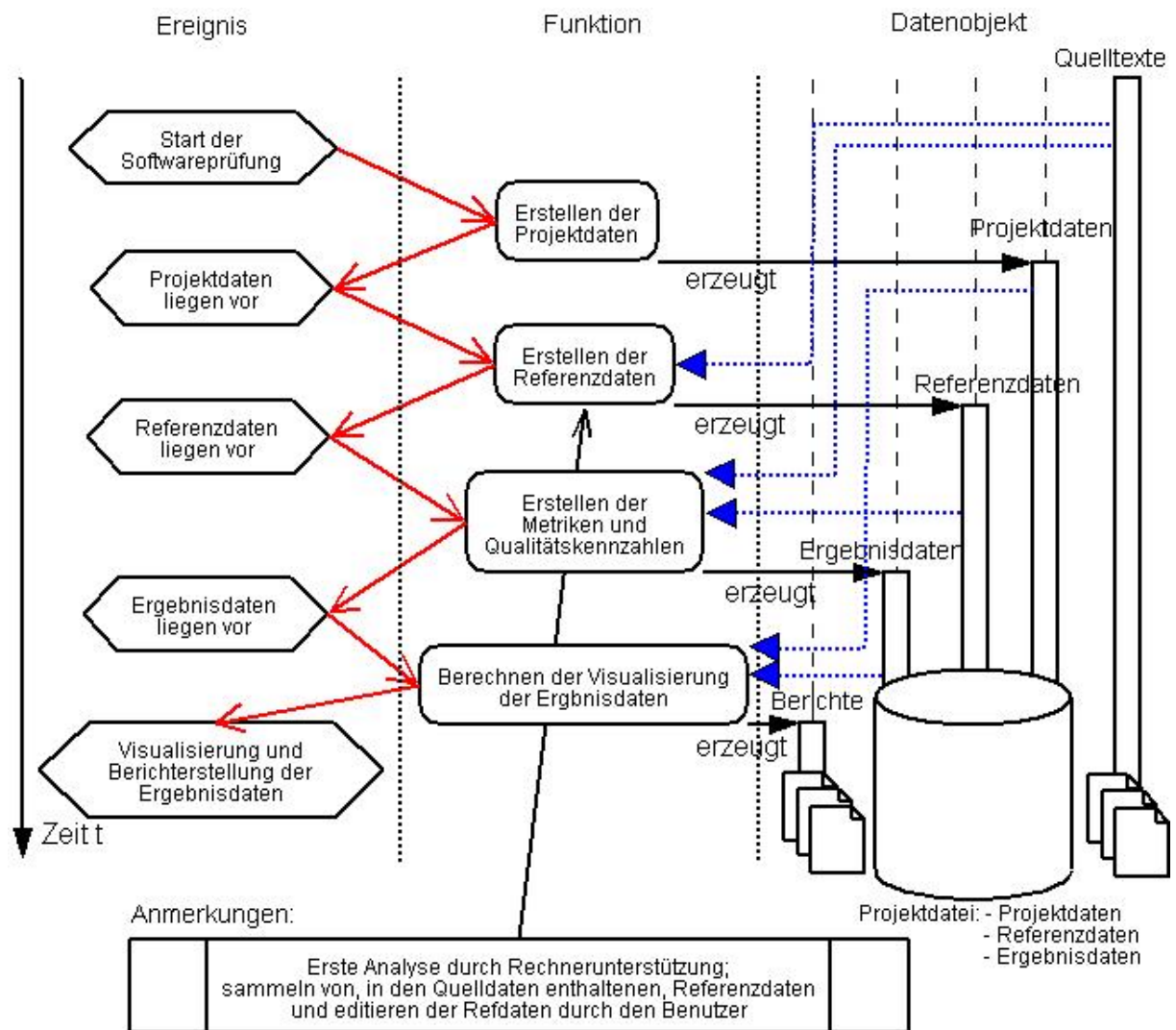


Abbildung 18 – Soll – Ablaufprozess eines zukünftigen Analysewerkzeugs

Die erheblichsten Änderungen beim Soll – Prozess gegenüber dem Ist - Zustand, betreffen die Verwaltung der beim Prozess generierten Daten und die Rechnerunterstützung bei der Konfiguration und Abwicklung des Gesamtprozesses. Zu den, während einer Prüfung anfallenden Daten gehören zum Beispiel nun auch Projektdatei, die bisher bei LOGISCOPE nicht betrachtet wurden. Solche könnten Projektname und -beginn, aber auch allgemeine Informationen zu den Quelltexten, Kunden- oder Bearbeiterdaten sein. Die Datenhaltung, in die alle Informationen eingehen, soll zukünftig einer Datenbank entsprechen und nicht mehr in verschieden strukturierten Klartextdateien erfolgen. Darüber ist sichergestellt, dass die Werte zentral und strukturiert vorliegen, was eine zyklische Datensicherung, die Erstellung von Berichten und Analysen auf Basis der Daten und Erweiterungen des Datenbe

stands erleichtert. Die Datenbank liefert die erforderlichen Werte für die Berechnung der Metriken und der graphischen Darstellungen und ist somit Datenbasis für weitere Prozessschritte. Angedacht ist außerdem eine Unterstützung bei der Verwaltung der Referenzdaten durch einen ersten Analysevorgang der zu evaluierenden Quellcodes. Dabei werden die in den Quellen enthaltenen Ausdrücke ausgewertet und bereits initial Klassen von Referenzdaten zugeordnet. Bei der weiteren Bearbeitung überprüft der Anwender die in dem ersten Durchlauf angelegten Referenzdaten auf ihre korrekte Klassifizierung und ändert sie gegebenenfalls.

Um die Ist – Architektur der Unix basierten Workstation abzulösen, wird eine Frontend – Backend – Aufteilung der Anwendung angestrebt, die die im vorigen Abschnitt angerissene Datenbankarchitektur unterstützt. Dabei wird das Frontend als „Fat“ – Client, mit der vollständigen Verarbeitungslogik und dem „Graphical User Interface“ (GUI) für den Anwender, entwickelt und für den Betrieb auf dem Arbeitsplatz der Benutzer installiert. Das Backend bildet eine Projektdatei, welche als Datenbank strukturiert die zentrale Datenhaltung implementiert. Bei der Nutzung des Analysewerkzeugs wird das Frontend mit den Tabellen des Backend für die Zeit der Verarbeitung verbunden, so dass auf die Daten zur Abfrage und Manipulation zugegriffen werden kann. Dies ermöglicht die Erhöhung der Robustheit und Skalierbarkeit der Gesamtanwendung, sowie eine Steigerung der Effizienz bei der Bearbeitung gegenüber der Workstation Architektur. Das Konzept ist in der Literatur unter der Bezeichnung des „zweischichtigen Client – Server - Modells“ bekannt. [Sommerville 2003, S. 255]

Wie die folgende Abbildung zeigt, kann bei dieser Frontend – Backend – Architektur die Projektdatei auf einem Server im Netzwerk oder lokal auf dem Client des Gutachters liegen.

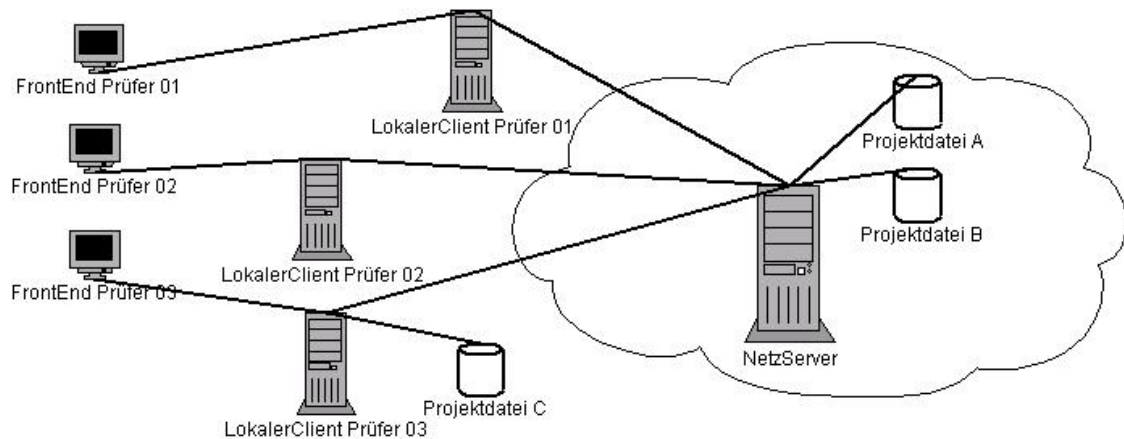


Abbildung 19 – Softwareverteilungsmuster des zukünftigen Analysewerkzeugs

In der bisher eingesetzten Anwendung wird die Berichterstellung und graphische Darstellung allein im Analysewerkzeug realisiert und lässt sich aufgrund seiner Struktur nicht mit gängigen Büroanwendungen über beispielsweise Programmierschnittstellen koppeln beziehungsweise ausbauen. Daher wird für die Individualentwicklung eine offene Programmierschnittstelle bevorzugt, die die geschilderten Mankos der LOGISCOPE Applikation durch die freie Kopplung der Neuentwicklung an standardisierte Werkzeuge erlaubt. Somit können zum Beispiel Berichte von Gutachtern mit Marken für Prüfungen und Reviews versehen und die Dokumente via Kommunikationsstrukturen verteilt werden.

5.3 Softwaredesign – Grobentwurf

Um die Eigenschaften der Applikationsstruktur detaillierter zu spezifizieren und konzipieren, werden folgend verschiedene Betrachtungen auf die Individualsoftware anhand UML modelliert.

5.3.1 Analyse der Anwendungsfälle und des Gesamtprozesses

Der Anwender des Analysewerkzeugs ist typischerweise der Prüfer des Softwareprodukts. Unter dem ersten Punkt „Projektverwaltung“ sind alle Funktionen zusammengefasst, die zu Beginn des Gesamtprozesses vom Gutachter abzuarbeiten sind. Der Prüfer soll seine angelegten Prüfprojekte verwalten können, so dass er, je nach dem welches er bearbeiten möchte, dieses öffnen und editieren kann. Dazu gehört das Bearbeiten der allgemeinen Projektdaten sowie das Zuordnen der zu untersuchenden Quelltexte zu dem Projekt. In einem zweiten Teilprozess verwaltet und konfiguriert er die Referenzdaten, die mit den zuvor zugeordneten Quelldatei

en als Grundlage zur Erstellung der Metriken dienen. Um zu einem späteren Zeitpunkt in der Verarbeitung die Qualitätskriterien berechnen zu können, werden in diesem Modul ebenso die Grenzwerte der Metriken angegeben und verwaltet. Mit Anstoß der Automatik zur Generierung der Kennzahlen startet die Verarbeitung. Die „Darstellung“ liefert Funktionen um die berechneten Metriken und Qualitätskriterien zu analysieren und auf den Ergebnissen aufsetzend Berichte und Abbildungen anhand anknüpfender Instrumente erstellen und bearbeiten zu können.

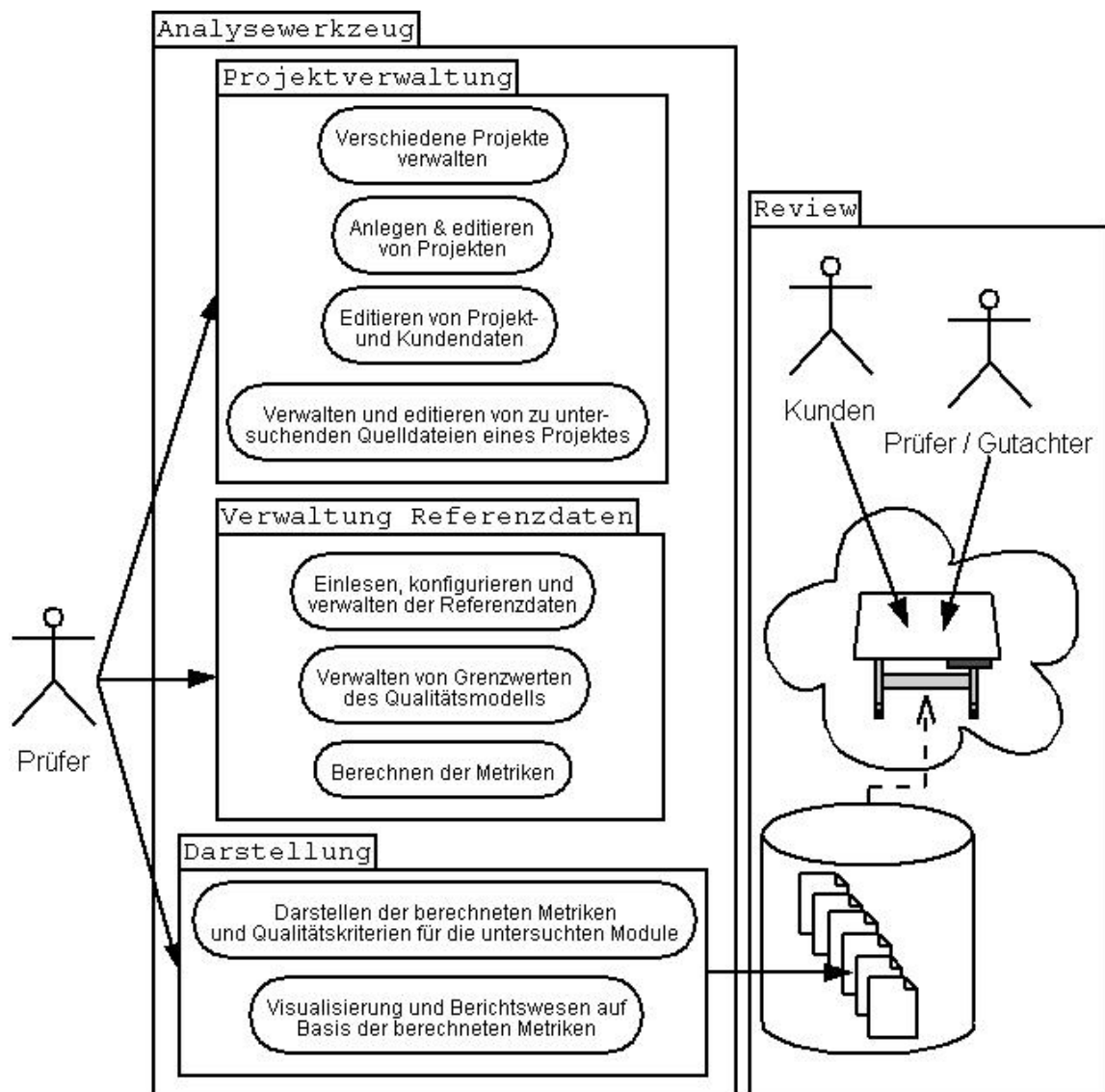


Abbildung 20 – Anwendung des zukünftigen Analysewerkzeugs

Wie in Abbildung 20 aufgezeigt, werden die Dokumente, die im letzten Anwendungsschritt des Analysewerkzeugs, der Komponente Darstellung, generiert werden, über eine Datenhaltung gesammelt und bilden die Grundlage für weitere A

analysen durch Kunden und Gutachter. Durch die Sammlung der Berichte in digitaler Form wird die zeitnahe Untersuchung durch mehrere, voneinander dezentral angeordneter, Instanzen ermöglicht. Über Kommunikationsstrukturen können diese geographisch unabhängig miteinander Analysen durchführen, was in der Illustration durch die „Wolke“ angedeutet wird.

Aufgrund der verschiedenen Anwendungsfälle, die durch die Verwendung des Analysewerkzeugs auftreten, lassen sich nach Aufgabentypen und Kontexten erste fachliche Gruppen von Aufgaben bilden. Gemäß der Darstellung in Abbildung 20, wird das Gesamtsystem nach den Subgruppen „Projektverwaltung“, „Verwaltung Referenzdaten“ und „Darstellung“ strukturiert. Diese Aufteilung wird sich durch den gesamten Entwicklungsprozess ziehen und somit in dieser Grundform erhalten bleiben.

5.3.2 Auf dem Soll – Konzept aufbauendes Ablaufmodell

Der sequentielle Ablauf der Applikation wird bereits durch die Darstellung in Abbildung 18 hinreichend visualisiert und bedarf daher keiner weiteren Abbildung durch ein UML – Sequenzdiagramm. Beim Ablauf werden Daten, wie die anfallenden Projekt- und Kundendaten, Referenzdaten und die Ergebnisdaten erzeugt. Anders als im bisherigen Prozess, werden diese über ein darauf angepasstes Datenhaltungskonzept gesammelt und verwaltet. Die graphischen Repräsentationen müssen nicht mehrfach bei Abruf generiert werden. Einmalig mit Anstoß der Verarbeitung, kann der Anwender die Darstellungen in Form von Berichten erzeugen. In einer Art Arbeitsmappe stehen sie den Nutzern danach immer wieder zur Verfügung und ermöglichen sogar die Individualisierung der Reporte anhand weiterführender Werkzeuge.

Analysedaten können zu jedem Zeitpunkt neu generiert werden, wobei zuvor berechnete Werte wenn sie veraltet sind aus der Datenhaltung gelöscht werden. Zum Beispiel würde dies bedeuten, dass Metriken und Kennzahlen gelöscht werden, wenn die Referenzdaten, die die Grundlage zur Berechnung dieser bilden, verändert werden.

5.3.3 Komponenten und Subsystemstruktur

Das Anwendungsfalldiagramm aus Abbildung 20, verdeutlicht die Einteilung der Verarbeitungslogik in die drei wesentlichen Bestandteile „Projektverwaltung“, „Verwaltung Referenzdaten“ und „Darstellung“. Darüber hinaus wird ein Bereich

benötigt, der dem Benutzer das Graphical – User – Interface zur Verfügung stellt. Über eine Schnittstelle zu einer weiteren Komponente werden alle im Prozess erzeugten Daten gesammelt. Einen weiteren Teilbereich bilden das Berichtswesen und die graphische Darstellung. Diese sollen zukünftig auch ohne die Bereitstellung des Analysewerkzeugs funktionsfähig sein und Funktionalitäten bereitstellen.

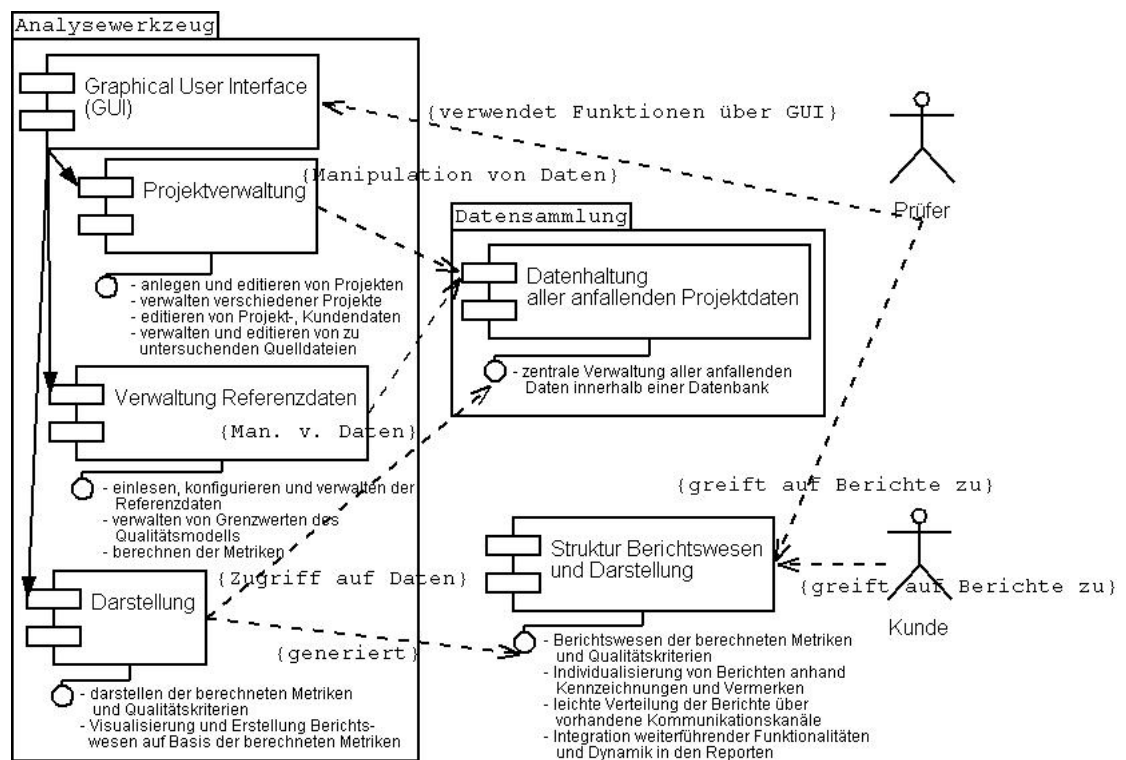


Abbildung 21 – Komponentendiagramm

Die GUI stellt dem Prüfer alle Funktionen des Analysewerkzeugs zur Verfügung. Intern manipulieren die Komponenten „Projektverwaltung“ und „Verwaltung Referenzdaten“ die Projektdaten, die vom Analysewerkzeug getrennt in der zentralen Datensammlung verwaltet werden. Die Komponente zur Darstellung und Visualisierung ist die einzige, die lediglich die Inhalte der Datensammlung ausliest und nicht manipuliert. Sie benötigt die Daten zur Illustration im Analysewerkzeug, sowie zur Generierung und Berechnung der Berichte, die die Werte aus der Sammlung repräsentieren. Vom eigentlichen Analysewerkzeug getrennt, können Prüfer und Kunden gleichermaßen auf die Berichte zugreifen und deren Funktionen nutzen.

5.4 Softwaredesign – Feinentwurf und Auswahl der Programmierwerkzeuge

Die Aspekte der Grobentwürfe gilt es im Feinentwurf eine Stufe weiterzuentwickeln. Im Hinblick auf die Implementierung werden die zuvor spezifizierten Modelle und Komponenten auf ihre einzelnen Bestandteile und die verwendeten Technologien herunter gebrochen. Die auszuwählende Technik muss dabei stets im Kontext der in 4.4.3 definierten Anforderungen betrachtet werden.

5.4.1 Komponenten des Analysewerkzeugs

Nach Abbildung 21 entsprechen die Komponenten GUI, Projektverwaltung, Verwaltung Referenzdaten und Darstellung den wesentlichen Teilen des Analysewerkzeugs. In einer umsetzenden Technologie sollten sie daher zusammen als integrierte Einzelpakete betrachtet werden. Ferner muss die Technologie den in 4.4.3 formulierten Ansprüchen des BIA genügen, so dass die Applikation in einer zweischichtigen Client – Server – Architektur realisiert werden kann, die zudem vor allem eine vereinfachte Handhabbarkeit, einen hohen Grad an Rechnerunterstützung und eine, für Weiterentwicklungen benötigte offene Programmierschnittstelle im Sinne des Rapid Application Development (RAD) – Konzepts aufweist.

Die GUI entspricht in dieser Betrachtung der Schnittstelle zwischen den Einzel-funktionalitäten der weiteren Komponenten. Der Anwender greift ausschließlich über sie auf die Applikation und die darin implementierten Funktionen zu. Bei Umsetzung der Modelle wird das Analysewerkzeug als eine Anwendung betrachtet,

die unabhängig von der Komponente zur Datenhaltung und zur Datenstruktur des Berichtswesens ablauffähig ist und genutzt werden kann.

Zur Datenhaltung wird eine relationale Datenbank vorgesehen, die in einem Dateiformat eine Art Projektdatei widerspiegelt. Getreu dem Datenbankschema können Manipulationen und Leseoperationen über einen SQL – Dialekt effizient realisiert werden, so dass die Datenbankzugriffe strukturiert erfolgen. Darüber hinaus erleichtert diese Art der Datensammlung die Anknüpfung weiterer Analysen und Auswertungen auf Basis der Daten, beziehungsweise vereinfacht eine Erweiterung der zu verwaltenden Informationen. Das Prinzip der Projektdatei empfiehlt sich, um leicht Datensicherungsmechanismen zu integrieren und den Austausch zwischen prüfenden Instanzen voranzutreiben.

Die Struktur des Berichtswesens und der graphischen Darstellung bedingt die komfortable Handhabung. Dieser Aspekt sollte auch im Hinblick auf Verteilung via Kommunikationsinfrastrukturen und Analyse über dezentrale Standorte hinweg betrachtet werden. Über eine Programmierschnittstelle können Berichte als dynamische Objekte gestaltet werden, die eine funktionale Analyseunterstützung anhand von Werkzeugen realisieren. Die Verwendung von weiteren Hilfsmitteln, wie Zeichengeräten und Notizen, würde Prüfern die Möglichkeit bieten, digitale Vermerke in die Berichte einzubringen.

5.4.2 Auswahl der verwendeten Programmierwerkzeuge

Visual Basic for Application (VBA) ist eine Skriptsprache, die in einem Grossteil der Microsoft Office Produkte seit der Version Office '97, hinterlegt ist. Bestandteile von VBA sind die Programmierwerkzeuge und Bibliotheken von Visual Basic (VB) und weitere Objektbibliotheken, die spezielle Gebilde der einzelnen Office – Anwendungen repräsentieren. VBA wurde den Anwendern von Office – Produkten zur Entwicklung eigener Software auf Basis der Office – Anwendungen beziehungsweise zur Erweiterung vorhandener Funktionalitäten, seitens Microsoft zur Verfügung gestellt. Im Gegensatz zu VB – Programmen, werden VBA – Anwendungen nicht in direkt ausführbaren Code compiliert und sind daher nicht alleinig auf einem Rechner ablauffähig. Sie bleiben in ihrer Skriptsprache bestehen und werden von einer Laufzeitumgebung während der Verarbeitung interpretiert und ausgeführt. Dies bedingt, dass zum Ablauf eines VBA – Programms die Laufzeitumgebung auf dem System installiert sein muss. Beispielsweise würde eine Office – Anwendung die Laufzeitumgebung stellen, in der dann der VBA – Code verarbeitet und ausgeführt

wird. Die dabei nötige Interpretation der in VBA verwendeten Instruktionen zur Laufzeit beschränkt eine VBA – Anwendung hinsichtlich seiner Performanz gegenüber einem compilierten Programm. Dennoch beinhaltet VBA Vorteile, die von anderen Programmiersprachen häufig nicht erfüllt werden können. Durch die hohe Verbreitung von Microsoft Office – Produkten kann VBA vielseitig eingebunden werden und ist inzwischen zu einem etablierten Standard in der Softwareentwicklung kleiner Büroapplikationen geworden. Die Entwicklung von VBA – Programmen benötigt darüber hinaus keinerlei weitere Installationen auf dem Rechnersystem, da viele der Office – Anwendungen VBA und die dazugehörige Entwicklungsumgebung integrieren. Besteht der Bedarf eine bessere Performanz einer VBA – Applikationen zu erzielen oder VBA – Programme ohne Office – Anwendungen als Laufzeitumgebung ablaufen zu lassen, lässt sich der Code in VB übertragen und daraus zu einem selbst ablauffähigen Programm compilieren. [Born 1999, S. XV, 59]

Folgend werden exemplarisch einige Aspekte hinsichtlich einer VBA – Realisierung aufgeführt, welche eine VBA – Implementierung befürworten.

- Durch den Ausbau der PC – Arbeitsplatz – und Software – Infrastruktur im BIA ist ein hoher Verbreitungsgrad von Microsoft Office – Anwendungen gewährleistet. Die Individualsoftware könnte somit ohne zusätzlichen Installationsaufwand an den Arbeitsplätzen der BIA – Gutachter installiert werden. Plattformunabhängigkeit, wie sie beispielsweise durch die Programmiersprache Java geleistet wird, ist durch den weitreichenden Einsatz von Microsoftanwendungen auf Clientarbeitsplätzen im BIA kein Musskriterium zur Auswahl einer Programmiertechnologie. Daher wird dieses Kriterium für weitere Betrachtungen irrelevant und nicht näher beleuchtet.
- Weiterentwicklungen benötigen ebenfalls keine zusätzlichen Installationen, sondern können anhand der in der Office – Applikation integrierten Entwicklungsumgebung durchgeführt werden.
- Da es sich um eine Skriptsprache handelt, ist sie selbst für Unerfahrene schnell und intuitiv erlernbar, so dass die Verwendung von VBA schnell zu Lösungsansätzen und Implementierungen führt. Darüber hinaus ist festzuhalten, dass VB im BIA sehr verbreitet ist und vielseitig zum Einsatz kommt. Die Weiterentwicklung und Wartung einer VBA – Applikation kann daher leichter von Entwicklern durchgeführt werden, als unter Verwendung eines Sprachkonzepts, welches in einem ersten Schritt die Einarbeitung von

Entwicklern erfordert. Ferner ist ebenso nicht zu vernachlässigen, dass beim Erstentwickler weitreichende Kenntnisse bezüglich VBA vorherrschen, die die Prototypenentwicklung vereinfachen.

- Durch eine Umsetzung auf Basis von Microsoft Access können umfangreiche GUI – Entwicklungswerkzeuge und eine komfortable Implementierung der Verarbeitungslogik genutzt werden. Darüber hinaus ermöglicht Access eine einfache Anbindung an Datenbankmanagementsysteme, wahlweise über einen SQL – Dialekt oder spezielle VBA – Kommandos. Diese Trennung der Fachlogik und der Datenhaltung ermöglicht die erforderliche zweistufige Client – Server – Struktur mit einem Fat – Client als Frontend und einer Datenbank als Backend.
- Microsoft Access stellt ein Datenbankmanagementsystem mit darunter liegender relationaler Datenbank zur Verfügung. Über dieses kann die Projektdatei mit der zentralen Datenhaltung umgesetzt werden. Würden die Anforderungen zukünftig an die Datenbank steigen, so dass viele tausend Datensätze verwaltet werden müssten und die Access – Datenbank nicht ausreichend Potential aufweist, kann durch die zweischichtige Anwendungsarchitektur auf eine leistungsstärkere Lösung immigriert werden. Nach bisherigen Einschätzungen wird jedoch ein derartiges Datenbankpotential für die Applikation nicht benötigt.
- Durch das VBA zugrunde liegende Konzept der allgemeinen und der speziellen Programmbibliotheken von Office – Produkten, lassen sich eben solche Objekte leicht ansteuern und verwenden, was eine effiziente Automatisierung zwischen verschiedenen Applikationen ermöglicht. Besonders im Hinblick auf das Berichtswesen und die graphische Darstellung können Funktionalitäten anderer Anwendungen, wie Microsoft Excel, Word und Powerpoint verwendet werden und so zu weitreichenden Funktionalitäten und Ausbaupotential führen.

6 Umsetzung des Softwaredesign – „mEtRIKA“

Die aus dem Feinentwurf entwickelten Ansätze der Applikationsimplementierung sind Anlass die Anwendung gemäß der spezifizierten Anforderungen auf Basis von Microsoft Access in der Version 2002 zu implementieren. Darüber hinaus werden die Werkzeuge SciTE in der Version 1.56 zur Darstellung von Quelltexten, Microsoft Excel in der Version 2002 zur Implementierung des Berichtswesens und Inno Setup – Version 3.0.6 zur Generierung eines mEtRIKA – Installationspakets verwendet. Weitere Informationen zu den Anwendungen werden zu gegebenen Zeitpunkten in der folgenden detaillierten Dokumentation aufgeführt.

6.1 Microsoft Access, VBA und grundlegende Anwendungsarchitektur

Wie bereits in 5.4.2 erläutert, ist die Skriptsprache Visual Basic for Application (VBA) in der Installation von Microsoft Access enthalten und integraler Bestandteil zur Entwicklung eigener Software oder zum Ausbau vorhandener Funktionalitäten. Wurde eine Access – Datei angelegt, kann über die Menüleiste oder die Tastenkombination „Alt + F11“ die VBA – Entwicklungsumgebung gestartet werden.

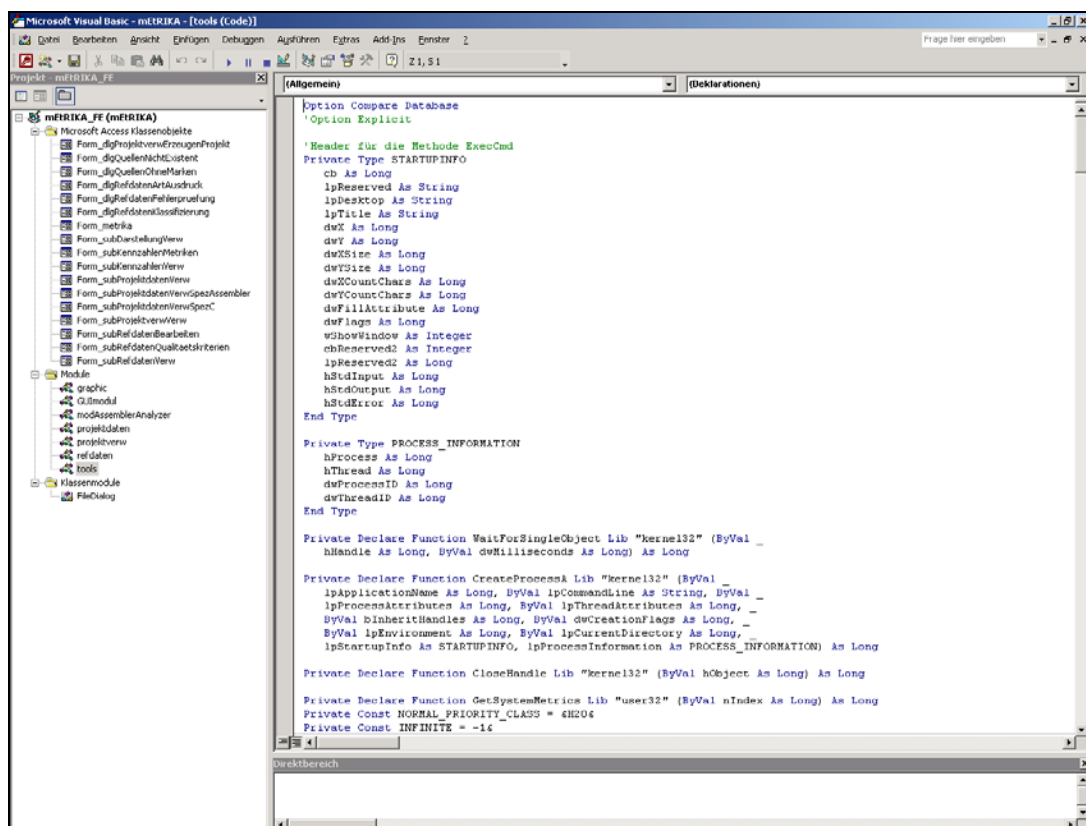


Abbildung 22 – VBA Entwicklungsumgebung

Bei der Softwareerstellung auf Basis von Microsoft Access und VBA, wird zwischen den Objekten „Microsoft Access Klassenobjekte“, „Module“ und „Klassenmodule“ unterschieden. Lediglich erstere sind spezielle Access – Gebilde und daher nur in VBA – Projekten aufsetzend auf Microsoft Access vorhanden. Die anderen Objektarten sind Bestandteil aller VBA – Projekte, unabhängig von der basierenden Laufzeitumgebung. Über Klassenmodule lassen sich VBA – Projekte entsprechend dem objektorientierten Paradigma entwickeln. Unter Modulen können Funktionen zusammengefasst werden, die sinngemäß einer Komponente zugehören. Die speziellen Microsoft Access Klassenobjekte bilden in unserem Entwicklungskontext die Formulare, die als Graphical User Interface dem Anwender zur Verfügung stehen. In allen drei Objektarten lässt sich VBA – Code in Deklarationen, Typdefinitionen, Funktionen oder so genannten Subs integrieren. Funktionsaufrufe erfolgen ebenso über alle Objektarten hinweg durch den Namen des Objekts, gefolgt von einem „.“ und dem Namen der Methode.

Eine Vielzahl weiterer Objekte lassen sich über Bibliotheken in die Anwendung einbinden. Über den Menüpunkt „Extras => Verweise“ wird ein Dialogfenster verfügbar, durch das, Bibliothekenverweise verwaltet werden können.

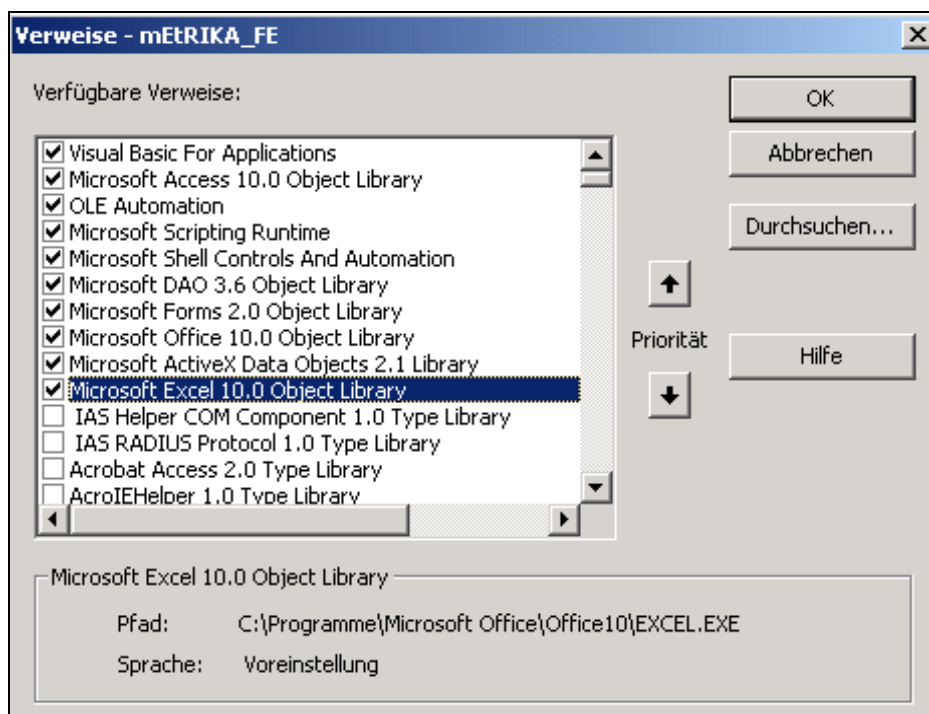


Abbildung 23 – VBA Bibliotheksverweise

Während der Ausführung des VBA – Codes werden benötigte Bibliotheken über die Verweise automatisch dem Projekt hinzugefügt. Fehlen Bibliotheken auf dem Rechnersystem, wird der Programmfluss an dieser Stelle gestoppt, so dass das Programm in seinem Ablauf unterbrocht und dem Benutzer den Mangel anhand einer Fehlermeldung darstellt.

Alle aktuell dem Programm zur Verfügung stehenden Objekte können über den Objektkatalog (Aktivierung über Menüleiste oder Taste „F2“ in der Entwicklungsumgebung) angezeigt werden.

Über die herkömmliche Access – GUI lassen sich Objekte wie Formulare und Module, auf die in der VBA – Entwicklungsumgebung zugegriffen werden kann, anlegen. Außerdem stehen weitere Objekte, wie Tabellen, Abfragen, Berichte, Seiten und Makros zur Verfügung.

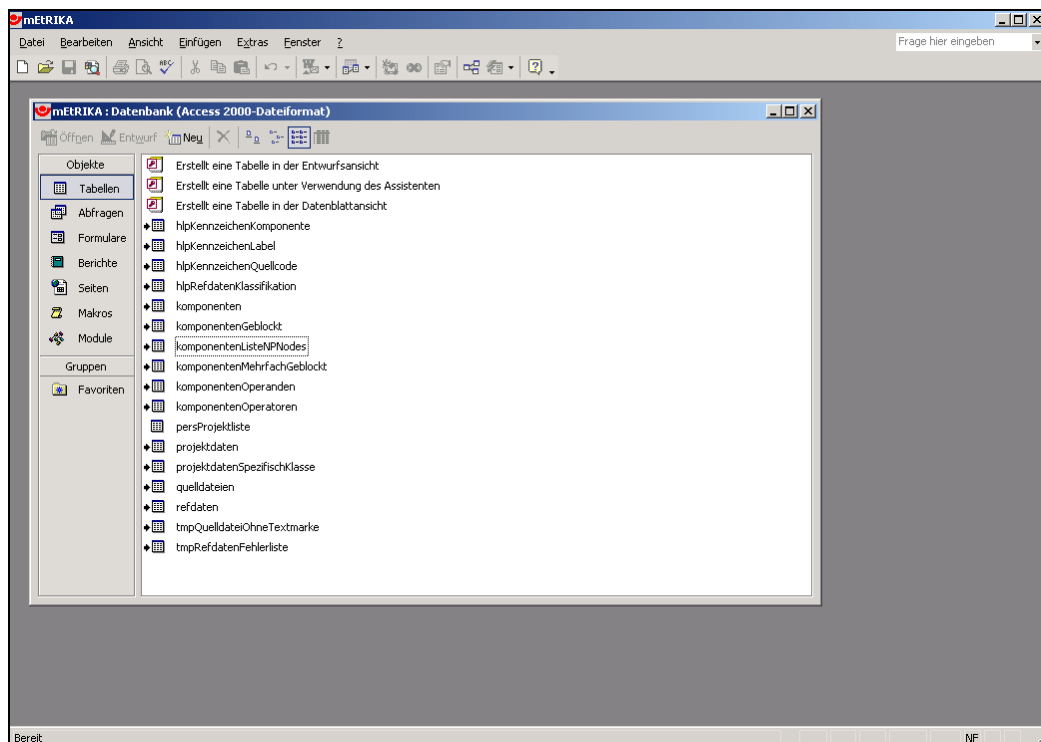


Abbildung 24 – Microsoft Access GUI

Um die, nach Spezifikation geforderte, zweischichtige Client – Server – Struktur zu realisieren, wird die Anwendung Frontend – Backend – aufgeteilt. Beide Komponenten werden in einer ersten Umsetzung über Microsoft Access implementiert. Demnach bildet das Frontend eine Access – Datei, die als Clientanwendung auf dem Arbeitsplatz des Anwenders installiert wird. Der zweite Teil, das Backend, wird durch

eine Access – Datei mit ausschließlich Tabellen gebildet. Diese Datenbankdatei entspricht dem relationalen Konzept und ist einzig für die Verwaltung der Daten verantwortlich (Kapitel 6.3).

In der zu entwickelnden prototypischen Version der Anwendung wird der Fokus auf zu untersuchende Assemblercodes gelegt. Die Applikation soll in der Lage sein, linear arbeitende Assemblercodes, wie beispielsweise die Befehlssätze der 80C51 – Derivate zu analysieren. Assemblerdialekte neuartiger Mikroprozessorfamilien, wie zum Beispiel die Sprachen des TMS 320C6x oder des TI C6x, die parallel, beziehungsweise konditional abgearbeitet werden, werden in dieser ersten Implementierung nicht für Untersuchungen eingeplant. Analysen von C – Programmen werden ebenso für zukünftige Betrachtungen eingeplant aber nicht implementiert. Weitere zukünftige Ansätze werden in der Zusammenfassung (Kapitel 7) betrachtet und daher in diesem und den folgenden Abschnitten dieses Kapitels nicht weiter beleuchtet.

6.2 Implementierung der Komponenten des Analysewerkzeugs

Um die einfache und intuitive Handhabung der Anwendung zu gewährleisten, wurde sie entsprechend dem Konzept eines Assistenten entwickelt. Bei Nutzung der Applikation orientiert sich der Anwender an den, im Hauptformular links angeordneten, Hauptfunktionen. Auf diese Weise werden alle relevanten Punkte durchexerziert und Basisdaten für Folgefunktionen generiert.

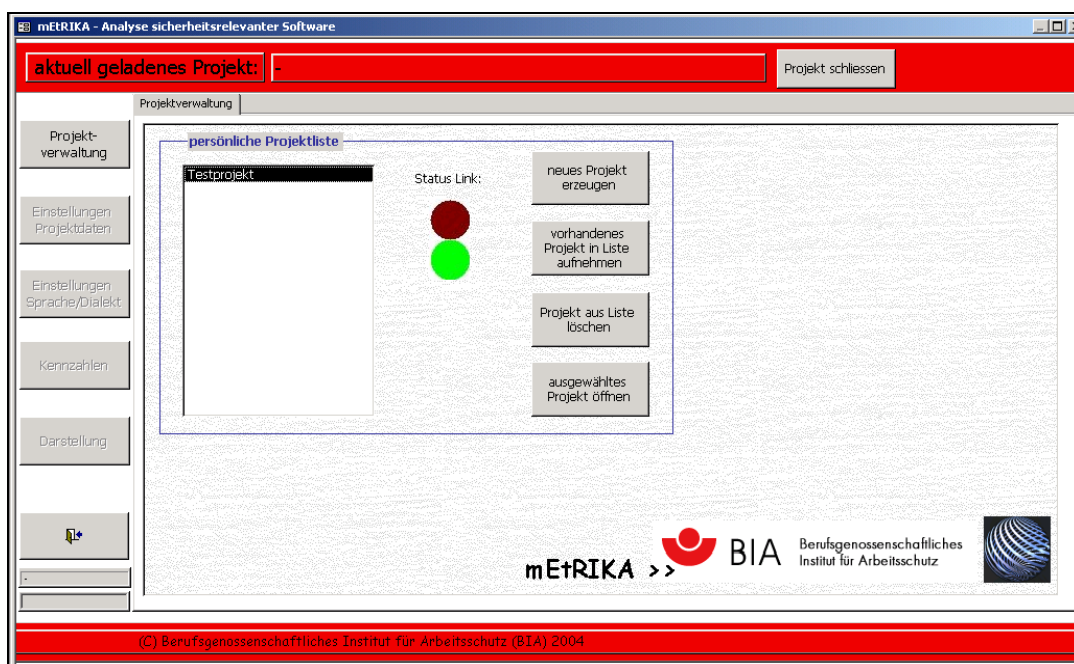


Abbildung 25 – Hauptformular mEtRIKA

Entsprechend den Hauptfunktionen aus Abbildung 25 (linke Seite) wurden die zu implementierenden Funktionalitäten in Modulen gruppiert.

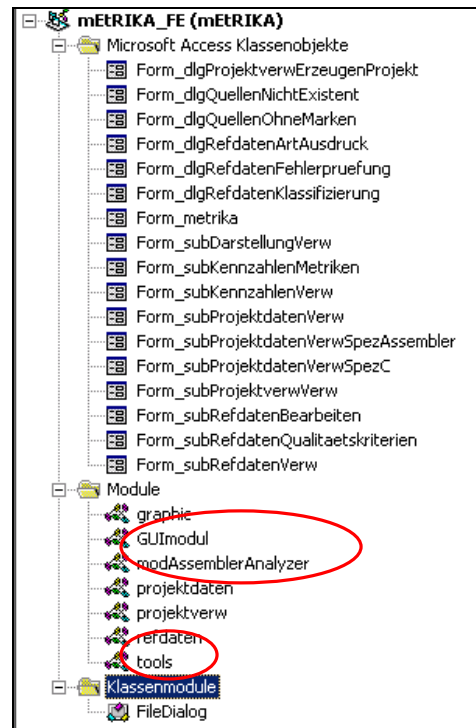


Abbildung 26 – mEtRIKA Modulstruktur

Die GUI wird über die Objekte der Microsoft Access Klassenobjekte und das zentrale Modul „GUImodul“ realisiert. Das Modul „tools“ definiert eine Reihe übergreifender Funktionen, die beispielsweise zur Stringverarbeitung oder zu Shell – Aufrufen und für Dateisystemoperationen benötigt werden. „modAssemblerAnalyzer“ enthält VBA – Quelltexte, die speziell auf die Untersuchung von Assemblercodes zugeschnitten sind, so dass alle bisher relevanten Analysefunktionen in diesem Modul enthalten sind. Eine Erweiterung der Applikation bezüglich eines Moduls, das beispielsweise C – Quellen analysiert, bedingt eine Implementierung dieser Assemblermethoden entsprechend für C – Programme.

6.2.1 Graphical User Interface – mEtRIKA

Über die Microsoft Access Klassenobjekte werden die GUI – Formulare der Applikation realisiert. In die Formulare eingebettet, befindet sich der VBA – Code, welcher zur Ablaufsteuerung der Verarbeitung benötigt wird. Auf diese Weise werden

beispielsweise Fehler abgefangen und die Analyse der Quelltexte gestartet. Der größte Teil der implementierten Subs und Funktionen bezieht sich auf die in den Formularen eingebetteten Steuerelemente, wie beispielsweise Buttons und Listen, und implementiert so deren Funktionalität.

Das „GUImodul“ im VBA – Code liefert Funktionen, die die Darstellung der Formulare im Kontext der Anwendung überprüfen. Beispielsweise sind einzelne GUI – Elemente, je nach Zustand der Verarbeitung, gesperrt, ein – oder ausgeblendet.

Eine Besonderheit der GUI sind die Funktionsaufrufe des SciTE Editors. Er schafft die unmittelbare Verbindung zwischen Analysemechanismen und den zugrunde liegenden Quelltexten. An vielen Stellen lässt sich durch den Editor der Quellcode über einen Funktionsaufruf aus dem Analysewerkzeug sichten. Sind dabei bereits Daten in der Referenzkonfiguration des Analyseprojekts enthalten, stellt der Editor die Quelltexte in markierter Form dar (Abbildung 27).

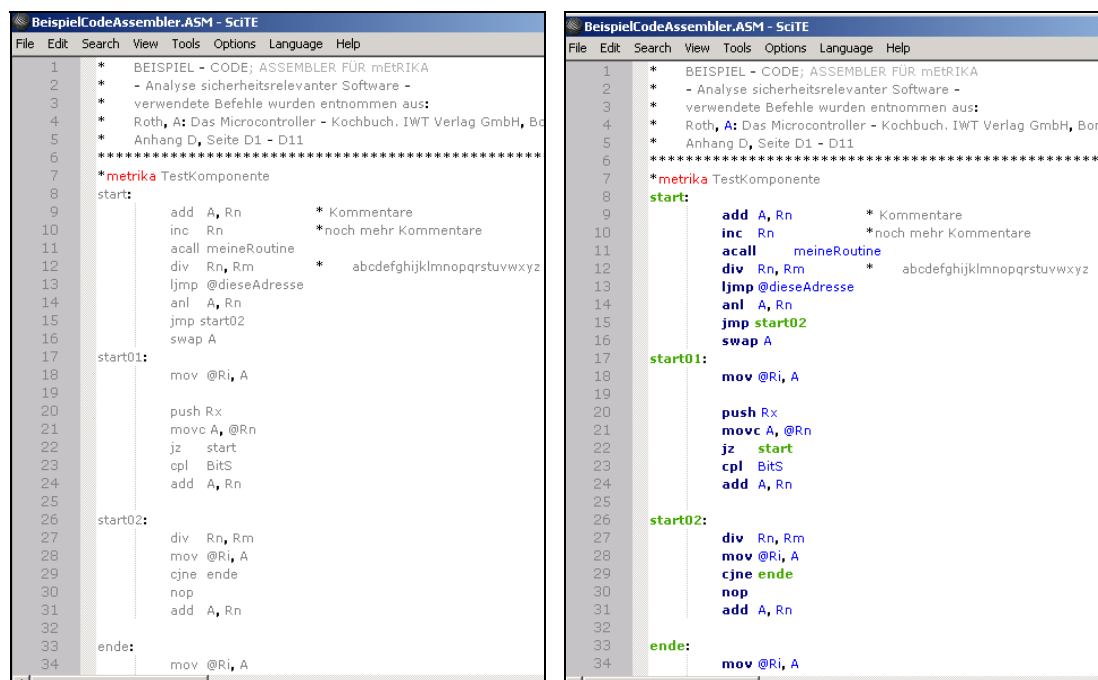


Abbildung 27 – SciTE – Editor mit Markierung der Referenzdaten

Die GUI – Struktur von mEtRIKA ist über ein stets sichtbares Hauptformular, in welches Unterformulare über ein Registersteuerelement eingebettet sind, implementiert (Abbildung 25). Je nach Benutzeraktion wird das entsprechende Unterformular sichtbar und bezüglich seiner enthaltenen Daten und Steuerelemente aktualisiert.

Zur Verwendung von Dateidialogen wurde nicht das standardisierte Steuerelement „Microsoft Common Dialog Control“ verwendet, sondern auf eine Eigenentwicklung von Karsten Pries zurückgegriffen. Dieses Modul wurde als Klassenmodul der VBA – Entwicklung beigefügt und ist damit auf jedem Clientarbeitsplatz in der Laufzeitumgebung verfügbar. Das typischerweise verwendete Steuerelement von Microsoft konnte nicht fehlerfrei in die Entwicklung eingebunden werden, so dass die ebenso standardkonforme Lösung von Karsten Pries die stabilere Variante bildet.

6.2.2 Projektverwaltung

In der Implementierung wird die Komponente Projektverwaltung über zwei Funktionsblöcke umgesetzt. Einerseits die „Projektverwaltung“ und zum anderen die „Einstellungen Projektdaten“. Über die Projektverwaltung wird die Verknüpfung des Frontend mEtRIKA mit der Projektdatei, die das Backend bildet, realisiert. Hierüber hat der Anwender die Möglichkeit mehrere Projektdateien in Form einer persönlichen Liste zu verwalten und ein ausgewähltes Projekt zur Bearbeitung zu öffnen. Existiert noch kein zu bearbeitendes Projekt, kann er ein neues erzeugen, welches dann selbstständig der persönlichen Projektliste hinzugefügt wird. Bei der Generierung eines Projekts ist dessen Typ festzulegen, welcher es als Assembler, beziehungsweise C Projekt in der späteren Verarbeitung identifiziert. Mit dem Öffnen des Projekts werden die Tabellen des Backend als Verknüpfung dem Frontend hinzugefügt und Bestandteil dessen.

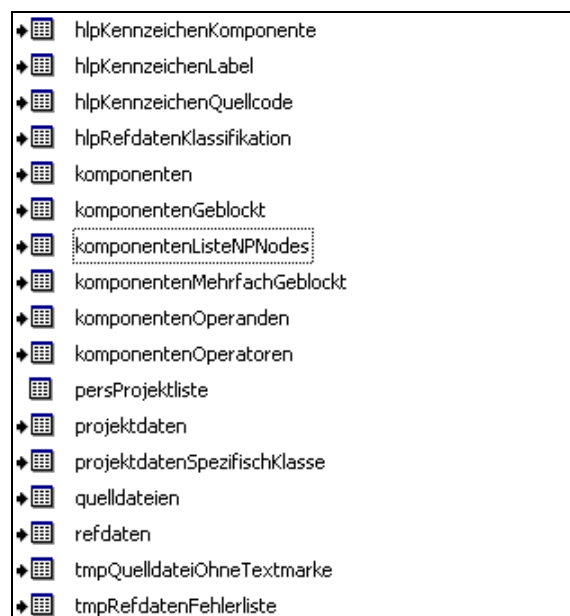


Abbildung 28 – Microsoft Access GUI – gelinkte Tabellen

Alle Operationen auf den Tabellen und den darin enthaltenen Datensätzen, können mit diesen Verknüpfungen programmtechnisch wie auf Tabellen des Frontend durchgeführt werden. Mit „schliessen“ der Projektdatei über die GUI, werden die verknüpften Tabellen aus dem Access – Frontend gelöscht, sind jedoch weiterhin im Backend, ergo der Projektdatei, verfügbar.

Der zweite Funktionsbereich von mEtRIKA bezüglich der Projektverwaltung bezieht sich auf die Handhabung von allgemeinen Projekt-, Kundendaten, projekttypspezifischen und Konfigurationsdaten. Projekttypspezifische Daten werden in Verbindung zum aktuell verknüpften Projekttypen betrachtet und betreffen die Einstellungen des relevanten Analysemoduls. Über die Konfiguration der Quelldateien können die zu untersuchenden Quelltexte logisch mit dem Projekt verbunden werden, so dass beim Analysevorgang die Applikation auf die Untersuchungsgegenstände zugreifen kann.

Die Realisierung der Komponente Projektverwaltung findet in mEtRIKA über die Module „projektverw“, „projektdaten“ und die entsprechenden GUI – Objekte statt. Erstes Modul implementiert alle Operationen, die zur Verwaltung von Projektdateien und deren Integration in das Frontend benötigt werden. „projektdaten“ enthält hingegen alle Funktionen, die Abfragen und Manipulationen der Backend – Daten vorsehen. Demnach sind nicht alle in „projektdaten“ enthaltenen Funktionen ausschließlich für die Projektverwaltung nutzbar, sondern bilden vielmehr eine Sammlung von Backend – Operationen.

6.2.3 Verwaltung der Referenzdaten und Berechnung der Kennzahlen

Anders als die Realisierung durch die Software LOGISCOPE, sieht mEtRIKA eine Rechnerunterstützung zur Erstellung und Verwaltung der Referenzdaten vor. Per Algorithmus werden in einem ersten Schritt die zu analysierenden Quelltexte untersucht und daraus relevante Referenzdaten ermittelt. Detaillierte Ausführungen zu diesem Mechanismus werden im folgenden Punkt 6.2.3.1 erläutert. Mit Generierung der ersten Referenzkonfiguration werden die erzeugten Daten Bestandteil der Konfiguration des SciTE – Editors. Dies ermöglicht, die Quelltexte mit besonderen Markierungen auf Basis der Referenzeinstellungen betrachten zu können und stellt somit eine wesentliche Erleichterung für die Betrachter der Quellen dar (Abbildung 27).

Über ein weiteres Formular können die Datensätze manuell anhand weitreichender Such- und Filterfunktionen geprüft und konfiguriert werden. Vor der Generierung der Metriken können ferner Einstellungen bezüglich der Grenzwerte von Qualitätskriterien vorgenommen werden. Diese werden ebenfalls über die Komponente Verwaltung Referenzdaten, beziehungsweise in der mEtRIKA – Applikation „Einstellungen Sprache / Dialekt“ abgebildet.

Die Berechnung der Metriken beginnt mit deren Anstoß im Anschluss an die Konfiguration. Eine umfangreiche Beschreibung des dafür implementierten Algorithmus liefert Punkt 6.2.3.2.

6.2.3.1 Generierung der Referenzdaten

Abbildung 29 zeigt den schematischen Ablauf der initialen Referenzdatenanalyse. Nach dem Anstoß der Verarbeitung wird innerhalb des GUI – Moduls geprüft, ob alle, für diesen Prozess benötigten Daten vollständig und korrekt angelegt sind, so dass die Verarbeitung auf sie zugreifen kann. Ist dies nicht der Fall, wird eine Meldung an den Benutzer generiert und die Verarbeitung gestoppt. Sind jedoch alle Daten verfügbar, wird die Methode „parsenRefdaten“ aus dem „modAssemblerAnalyzer“ – Modul gestartet. Innerhalb dieser Funktion werden bereits angelegte Daten, die durch den Start der Verarbeitung veralten, gelöscht und die Analyse der Quelldateien initiiert. Die wesentlichsten Bestandteile sind dabei, dass alle in einer Quelldatei vorhandenen Ausdrücke untersucht und als Referenzdatum aufgenommen werden. Lassen sich besondere Eigenschaften zu einem Ausdruck feststellen, wie beispielsweise, dass es der erste innerhalb einer Zeile ist und dahinter ein „:“ folgt, werden die Ausdrücke in die Klassen „Befehl“, „Label“ und „Parameter“ eingeteilt. Entsprechend dieser Klassifikation werden die Ausdrücke bei der Berechnung der Metriken gewertet. Als letzter Bestandteil der Funktion wurde eine „Sprungvorhersage“ implementiert, die prüft, ob ein Parameter eines Befehls einem Label entspricht. Ist dies der Fall, geht die erste Analyse davon aus, dass es sich bei dem Befehl um einen Sprung handelt, der mindestens „unbedingt“ ist. Aus diesem Grund wird dieser Befehl im ersten Analyseschritt als unbedingter Sprung klassifiziert, was wiederum die weitere Verwaltung der Befehle für die Gutachter vereinfacht.

Der Begriff der Komponente muss in Abbildung 29 und den folgenden Textpassagen zur Berechnung der Metriken im Kontext der Assemblercodes gewertet

werden. Eine Komponente entspricht dabei einer definierten Sequenz von Assemblerinstruktionen, ähnlich einer Funktion oder Methode einer Hochsprache.

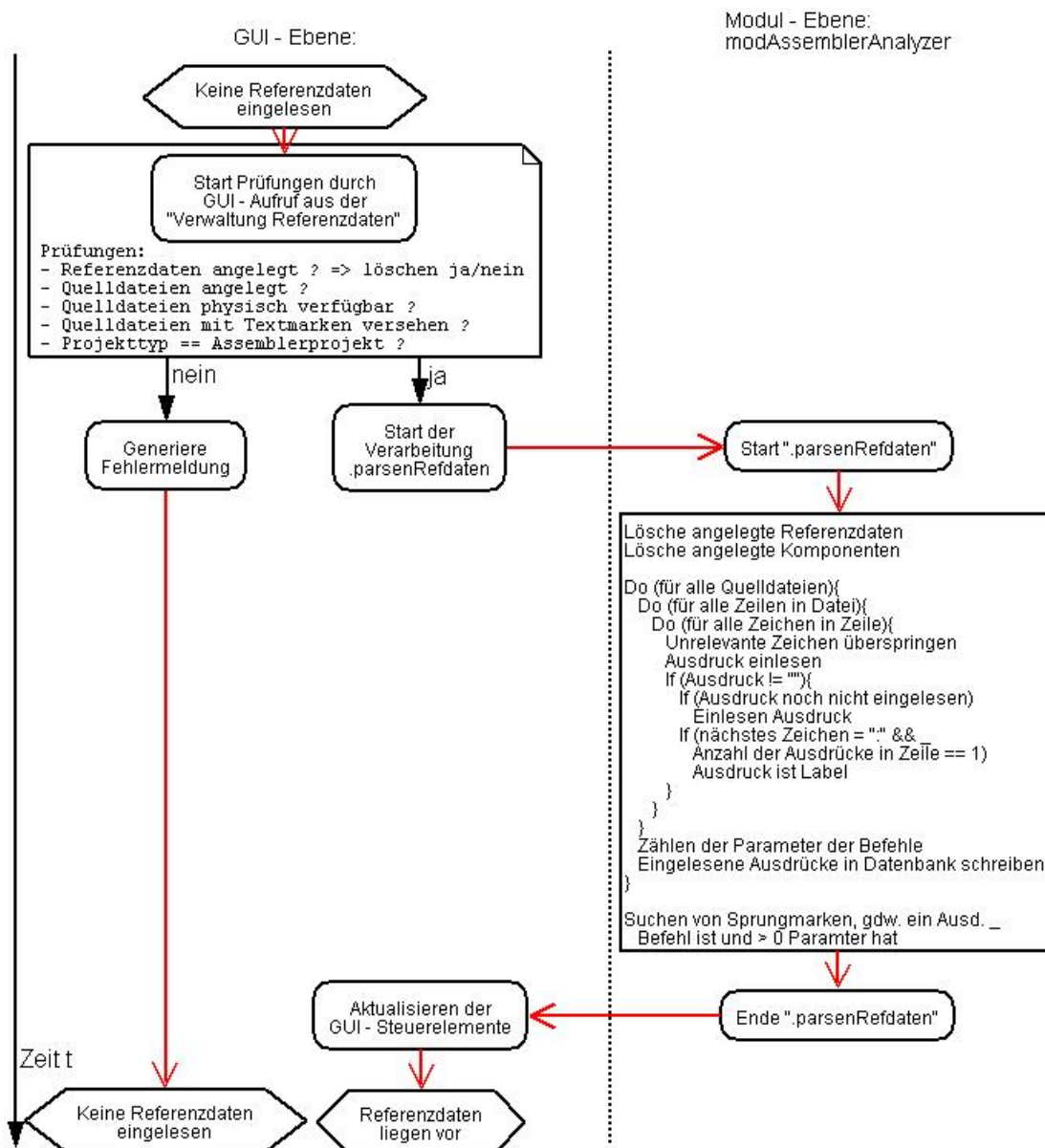


Abbildung 29 – mEtRIKA – Einlesen der Referenzdaten

6.2.3.2 Implementierte Metriken und deren Berechnung

Die in mEtRIKA umgesetzten Kennzahlen gleichen den Metriken der LOGSCOPE – Lösung und erweitern diese um die von Krell Entwickelten. Eine vollständige Liste der in mEtRIKA implementierten Metriken ist im Anhang - 1 zu finden. Dabei sind die aufgeführten Kennzahlen nach ihren Entwicklern geordnet. Ferner ist aus der Tabelle ersichtlich, welche Metriken den Synonymen der LOGSCOPE –

Implementierung beziehungsweise den Bezeichnungen von Krell entsprechen. Der letzte Listeneintrag zu jedem Datum zeigt auf, welche Metriken aus anderen berechnet werden und daher als „zusammengesetzte“ gelten, oder von der Programmlogik bei der Softwareanalyse ermittelt werden müssen. [Krell 2003]

Abbildung 30 zeigt die Struktur der Funktion `parsenMetriken()`, die in mEtRIKA zur Berechnung aller Kennzahlen ausgeführt wird.

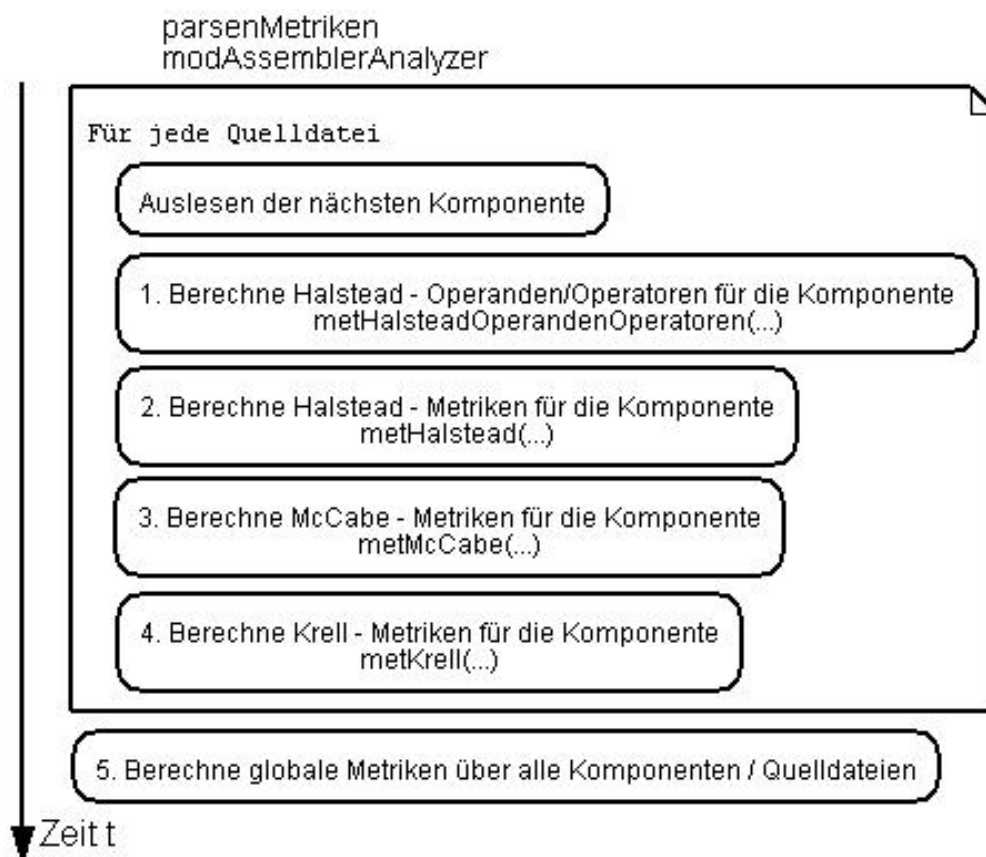


Abbildung 30 – mEtRIKA – Berechnung der Metriken

Für jede zu untersuchende Quelldatei werden dabei die Halstead-, McCabe- und Krell – Metriken berechnet und in der Projektdatei gesichert. Im letzten Schritt werden die globalen Metriken berechnet. Diese gehören zwar ebenso zu den McCabe- und Krell – Metriken, müssen jedoch bezüglich des Gesamtsystems und nicht nur einer isolierten Komponente zugehörig betrachtet werden.

Zu den Kennzahlen der „globalen Metriken“ gehören beispielsweise die Anzahl aller Eingänge in eine Komponente und die Anzahl aller nicht zyklischen Wege durch eine Komponente.

Über die folgenden Abbildungen werden die Teilfunktionen zur Berechnung der Metriken separat betrachtet und die Einzelschritte zur Berechnung dargestellt.

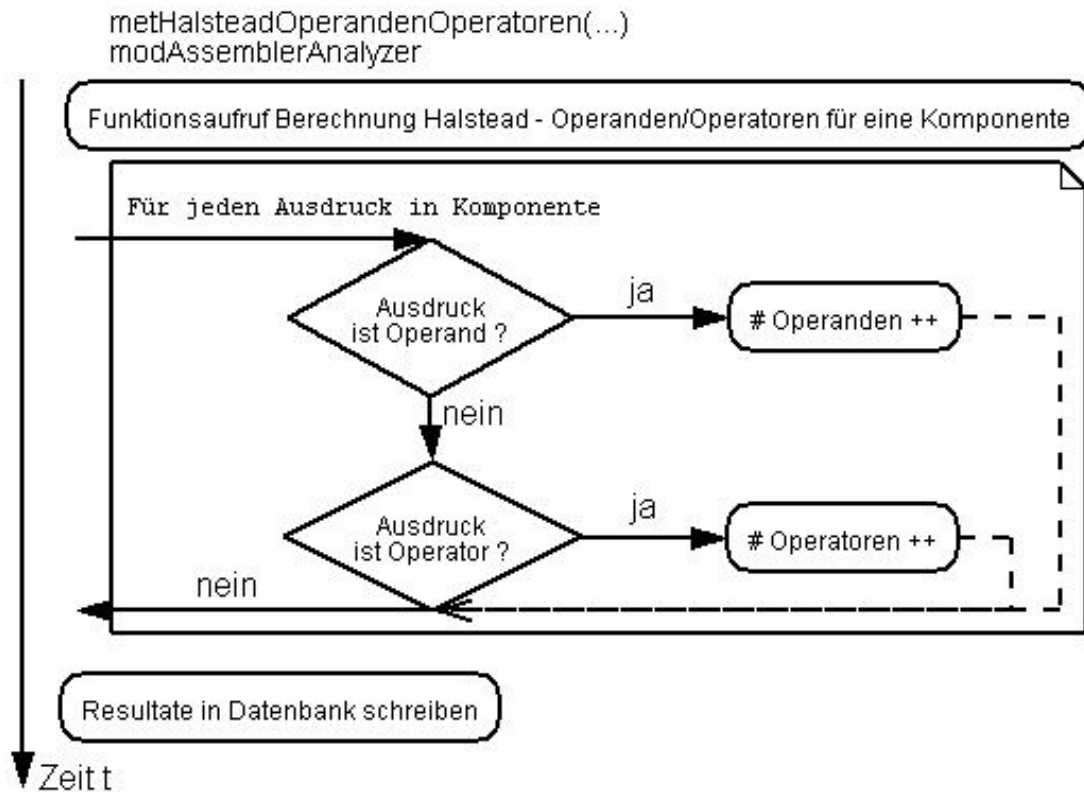


Abbildung 31 – mEtRIKA – Berechnung der Halstead – Operanden und Operatoren

Vor Generierung der Halstead – Metriken müssen die Operatoren und Operanden einer Komponente ermittelt werden. Nach den Definitionen dieser Codesegmente aus Kapitel 3.2.5, werden sie für jeden relevanten Ausdruck einer Komponente geprüft und gesammelt. Die Implementierung durch mEtRIKA definiert die Elemente im Aspekt der Referenzdaten folgend:

- Operatoren sind alle Ausdrücke, die als Referenzdatum der Klasse „Befehl“ angelegt wurden.
- Operanden entsprechen allen Ausdrücken, die als „Parameter“ oder „Label“ angelegt wurden. Ein „Label“ gilt jedoch nur als Operand, wenn es im Kontext des Quellcodes als Parameter hinter einem „Befehl“ auftaucht.

Mit Abschluss der Iteration werden die Resultate in die Projektdatei geschrieben, so dass sie späteren Berechnungen zur Verfügung stehen.

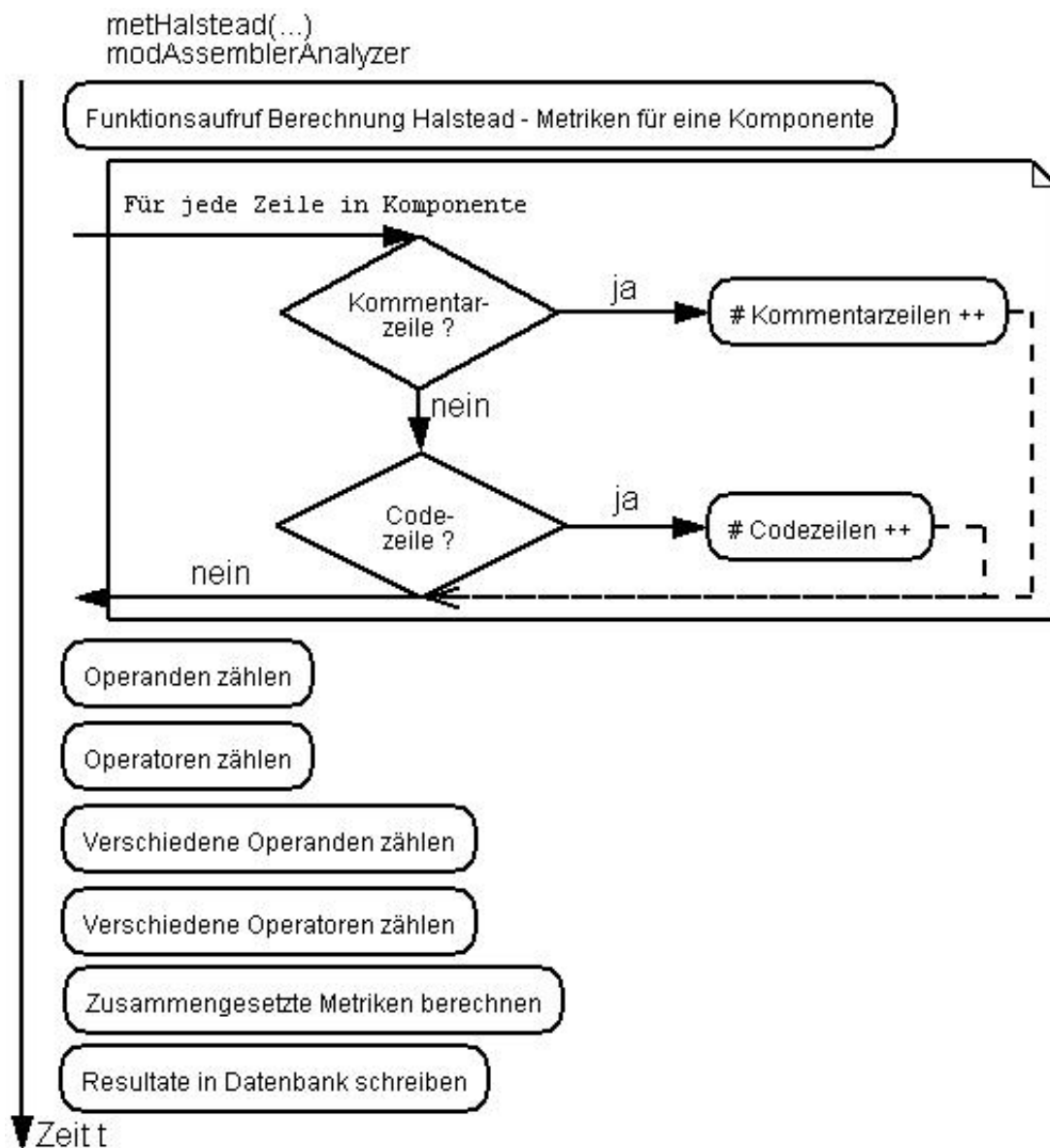


Abbildung 32 – mEtRIKA – Berechnung der Halstead – Metriken

Wurden die Operatoren und Operanden erfolgreich eingelesen, können die darauf aufsetzenden restlichen Halstead – Metriken berechnet werden. Um sie bestimmen zu können, muss lediglich die Anzahl der Operatoren, Operanden, der verschiedenen Operatoren und Operanden und die Anzahl der Kommentar- und Codezeilen gezählt werden. Alle weiteren Halstead – Metriken werden aus diesen sechs Grundmetriken zusammengesetzt und lassen sich direkt arithmetisch berechnen.

In der mEtRIKA – Umsetzung wird eine Kommentarzeile als Quelltextzeile definiert, die das „Kommentar einleitende Zeichen“ beinhaltet und dahinter mindestens fünf ASCII – Zeichen folgen, die nicht einem Leerzeichen entsprechen. Um eine Codezeile entsprechend der mEtRIKA – Definition zu haben, darf diese keine Leerzeile sein und muss einen Ausdruck aufweisen, der nicht als „Label“ klassifiziert ist.

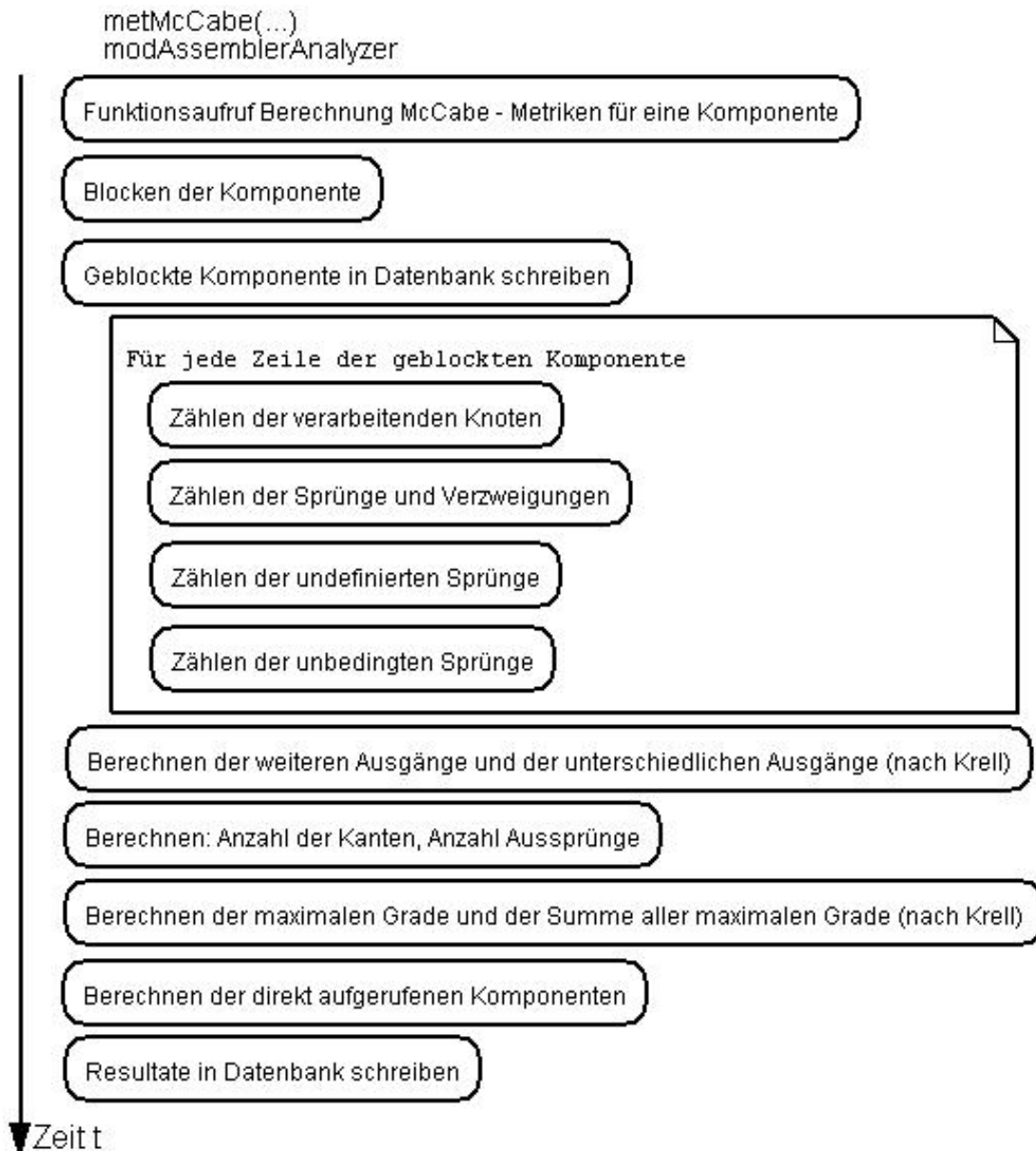


Abbildung 33 – mEtRIKA – Berechnung der McCabe – Metriken

Der Berechnung der McCabe – Metriken gilt der größte Aufwand, da ein Grossteil der Kennzahlen nicht arithmetisch berechnet werden können und daher für das

Ermitteln Iterationen nötig sind. Um die Strukturmetriken zu ermitteln, wird in einem ersten Schritt, wie in der LOGISCOPE – Verarbeitung zur Visualisierung, die zu untersuchende Komponente „geblockt“. Dabei wird eine Art „Pseudocode“, ähnlich dem der Anwendung LOGISCOPE (Kapitel 4.4.2), erstellt, der Grundlage für die Berechnung der Knoten und Kanten und somit die Basis für die Strukturmetriken darstellt. Beim Blockungsvorgang werden lineare Zeilen und Sprünge zu einzelnen Knoten zusammengefasst. Die Programmablaufstruktur wird durch Kanten zwischen den Knoten repräsentiert.

Da der Grossteil der dazugehörigen Metriken separat berechnet werden muss, sind viele Codeteile in eigene Funktionen eingeteilt um die Lesbarkeit der Gesamtfunktion zu erleichtern.

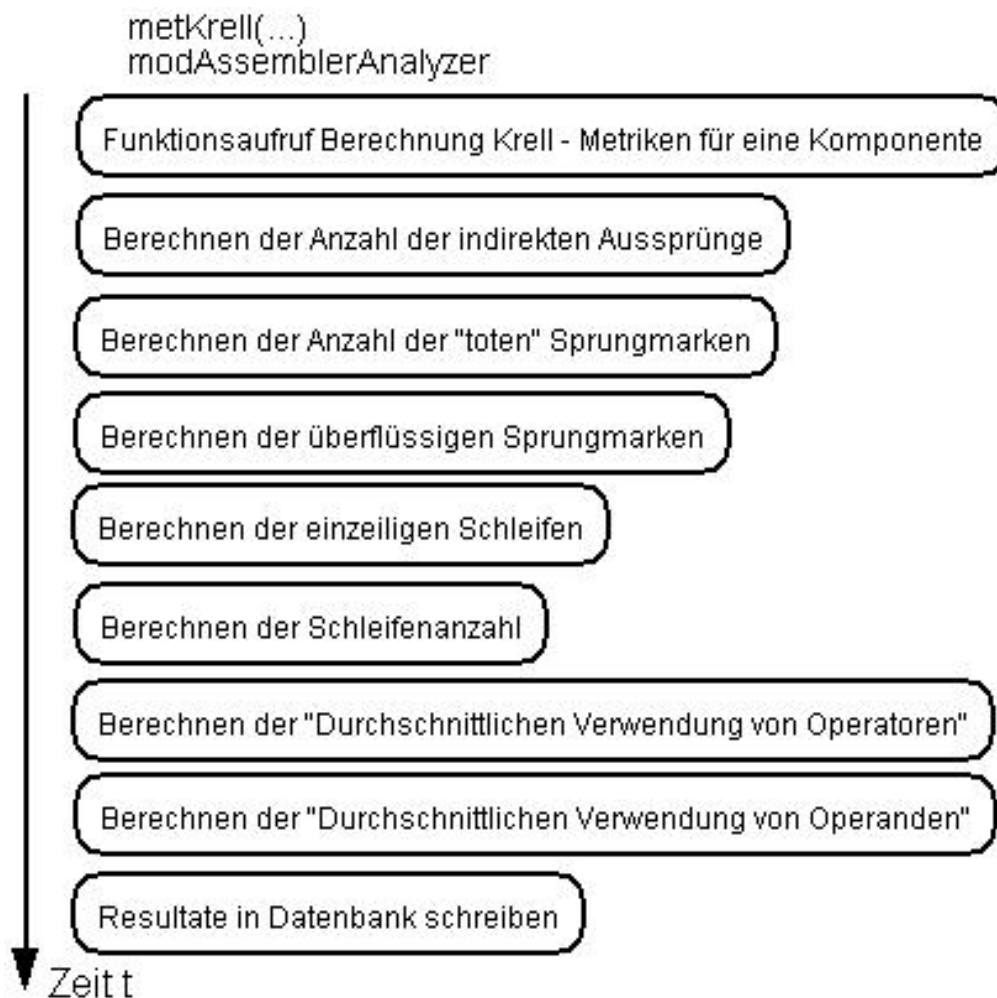


Abbildung 34 – mEtRIKA – Berechnung der Krell – Metriken

Ganz neue Betrachtungsspielräume bieten die Metriken von Krell, die nicht in der LOGISCOPE – Implementierung integriert werden konnten und daher zum ersten Mal bei einer Analyse berechnet werden können (Abbildung 34).



Abbildung 35 – mEtRIKA – Berechnung der „globalen“ – Metriken

Wie in der Einleitung dieses Abschnitts aufgezeigt, existieren unter den umgesetzten Metriken einige, die sich auf den Kontext des Gesamtsystems beziehen. Daher muss die Softwareanalyse auch das Gesamtsystem betrachten. Die Ermittlung dieser Kennzahlen ist innerhalb mEtRIKA unter der Funktion „metBerechnenGlobaleMetriken()“ (Abbildung 35) zusammengefasst und wird als letzte Funktion bei der Berechnung der Metriken ausgeführt.

6.2.4 Darstellung und Visualisierung der Kennzahlen

Wurden die Metriken berechnet, springt die Applikation auf die Darstellung der Kennzahlen. Der Bereich der Darstellung wurde in mEtRIKA über zwei Teile umgesetzt. Der Funktionsbereich „Kennzahlen“ übernimmt die Darstellung der Metriken anhand des Frontend selbst. Tabellarisch sind dabei die einzelnen Werte nach Metriktyp angeordnet. Typischerweise werden die Metriken in schwarzer Schrift dargestellt. Verletzt der Wert jedoch einen Grenzwert, der zuvor unter 6.2.3 konfi

gurierten Qualitätskriterien, wird er bei einer Unterschreitung blau und einer Überschreitung rot dargestellt.

Die Qualitätskriterien für eine Komponente lassen sich über ein weiteres Formular ablesen. Es stellt unterlegt dar, wie „gut“ eine Komponente bei der Berechnung der Testbarkeit, Selbstbeschreibung, Einfachheit, Lesbarkeit und der globalen Qualität abgeschnitten hat.

Um die Zahlenkolonnen aus dem Frontend in Form von Berichten darzustellen, können unter dem Unterformular „Darstellung“ „Berichtsmappen“ generiert werden. Dabei wird über VBA die Programmierschnittstelle von Microsoft Excel 2002 angesteuert um eine neue Arbeitsmappe in Form einer Excel – Applikation zu öffnen, die berechneten Kennzahlen in diese zu übertragen und darauf aufbauend eine graphische Visualisierung zu erzeugen. Die graphischen Repräsentationen werden innerhalb verschiedener Arbeitsblätter nach Komponenten getrennt aufgebaut, so dass die erzeugte Exceldatei einer „Berichtsmappe“ entspricht, in der auf jedem Blatt ein Einzelbericht zu einer Komponente zu finden ist. Das erste Blatt der Berichtsmappe dient als Deckblatt, in dem Projektdaten und Einstellungen zur Berichtsmappe eingetragen sind. Die Einstellungen betreffen beispielsweise den Editor, der verwendet werden kann, um aus den Einzelberichten heraus Codesequenzen aus den grundlegenden Quelltexten betrachten zu können.

Darüber hinaus können durch Abbildung des Berichtswesens unter Anwendung von standardisierten Büroanwendungen weitere Werkzeuge aus den Applikationen zur Individualisierung der Berichte verwendet werden. Ein Beispiel hierfür zeigt Abbildung 36. In dem Bericht konnten anhand weiterführender Zeichen- und Vermerkwerkzeuge Markierungen und Notizen in digitaler Form integriert werden.

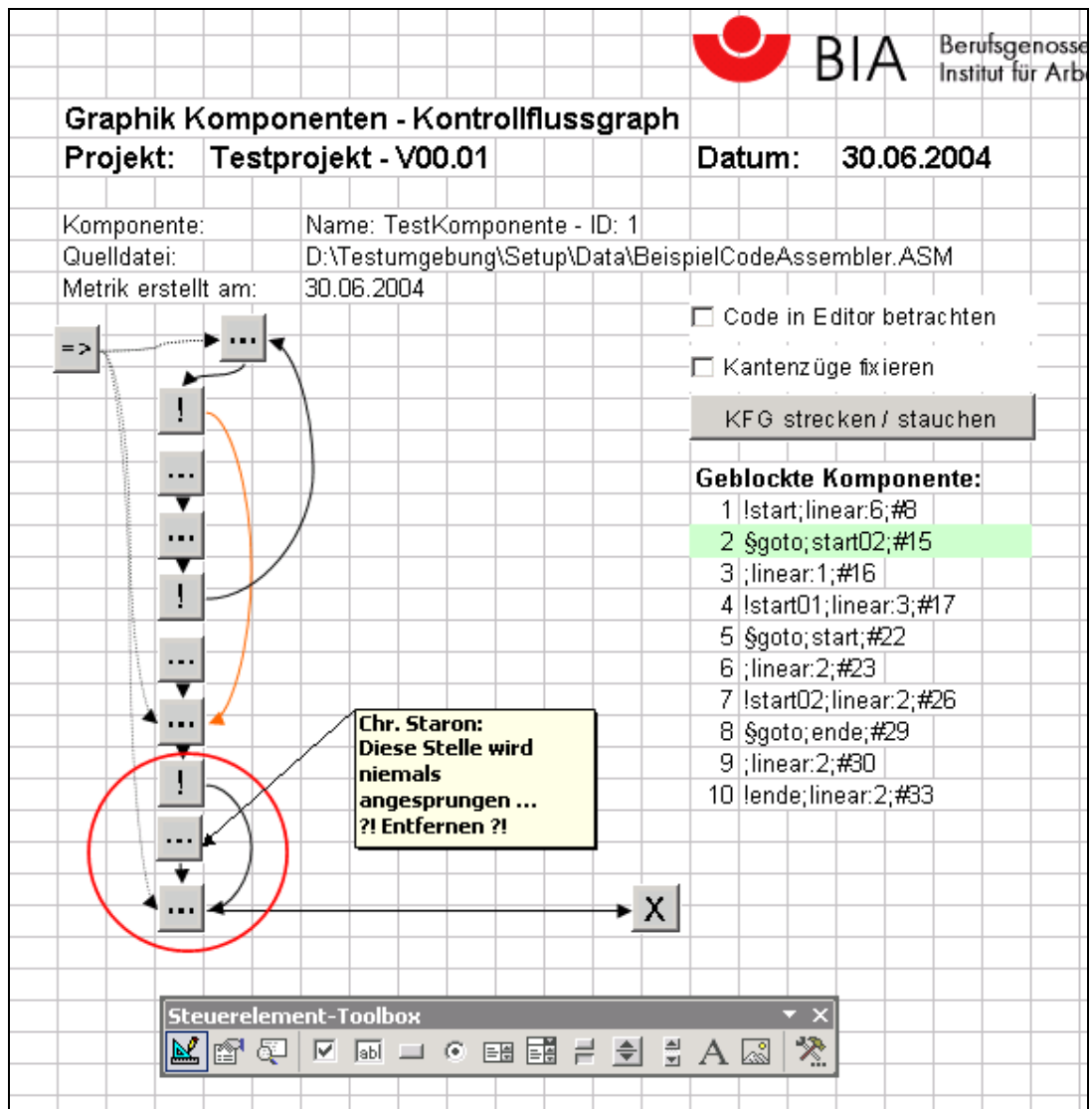


Abbildung 36 – mEtRIKA – Beispiel eines digitalen und individuellen Berichts

6.3 Datenhaltung und –verwaltung

Gemäß der verteilten Frontend – Backend – Architektur (Abbildung 19), die für mEtRIKA vorgesehen wurde, werden die Projektdaten in einem zentralen Backend gehalten und verwaltet. Durch die Implementierung der Applikation unter Verwendung von Microsoft Access 2002 kann die Datenhaltung als Projektdatei über das Access – Datenbank – Dateiformat „.mdb“ realisiert werden. Unter Nutzung des Frontend können Projektdateien auf dem Dateisystem mit allen dazugehörigen Tabellen erzeugt werden. Über die Funktion „projektverw.anlegenProjekt(...)“ wird in der mEtRIKA – Implementierung eine Datenbankdatei mit benötigten Tabellen und deren Einstellungen bezüglich Primärschlüsseln, Beziehungen und Initialisierungen generiert. Dazu können wahlweise VBA oder SQL – Befehle genutzt werden, die

unter Verwendung eines Datenbankobjektes an die Projektdatei abgesetzt werden können.

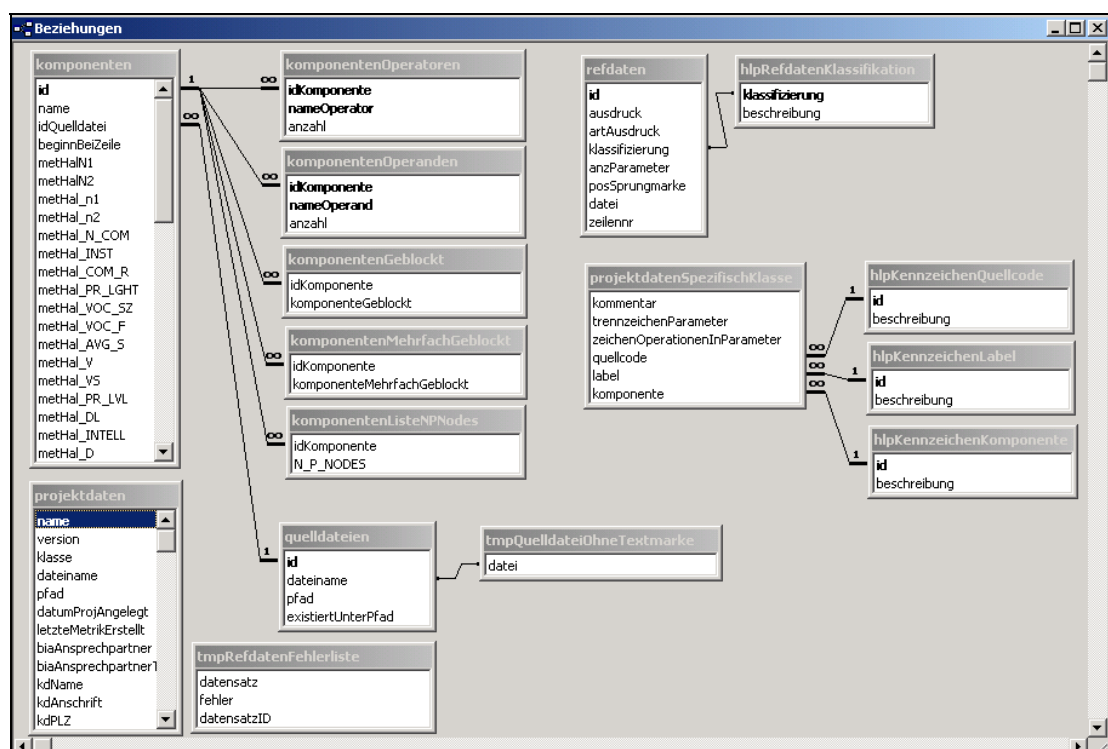


Abbildung 37 – mEtRIKA – Backend – Struktur

Wird auf die Projektdatei unter Verwendung des Frontend, beispielsweise bei der Bearbeitung einer Softwareuntersuchung zugegriffen, werden dafür beim „öffnen“ der Projektdatei die Tabellen des Backend mit dem Frontend verknüpft. (Abbildung 28)

6.4 Generierung des Auslieferungspakets – mEtRIKA und die Entwicklungsumgebung auf Verzeichnisebene

Die Analysesoftware mEtRIKA wird an die Anwenderarbeitsplätze als Installationspaket ausgeliefert. Unter Verwendung von Inno Setup in der Version 3.0.6 wird das Auslieferungspaket mit den Hauptbestandteilen mEtRIKA Frontend und dazugehöriges Benutzerhandbuch, SciTE – Editor und Data – Verzeichnis geschnürt. Das Data – Verzeichnis beinhaltet ein Assemblercode – Beispiel, welches zum Ausprobieren des Frontend geeignet ist, diverse Symboldateien und eine Exceldatei, die zur Generierung der Berichte benötigt werden.

Abbildung 38 zeigt die wesentlichen Bestandteile, die zur Generierung des Installationspakets benötigt werden. Das Verzeichnis „!SetupCompiler“ beinhaltet Inno Setup. Das Ausführen des Setupcompilers mit dem Setupskript „metrika.iss“ als Parameter (Beispiel eines Kommandozeilenaufrufs: „!SetupCompiler\iscc.exe Setup\metrika.iss“) startet die Verarbeitung. Der Compiler generiert daraufhin das Installationspaket nach den Konfigurationen des Setupskripts. Das Skript lässt sich durch einen Editor bearbeiten, so dass Einstellungen zum Installationspaket gemacht werden können. Teilschritte zur Generierung eines Setups, wie beispielsweise das Kopieren der aktuellen Frontend – Datei und des Handbuchs in das Setup – Verzeichnis, lassen sich in Form einer Windows – Batchverarbeitung automatisiert ausführen.

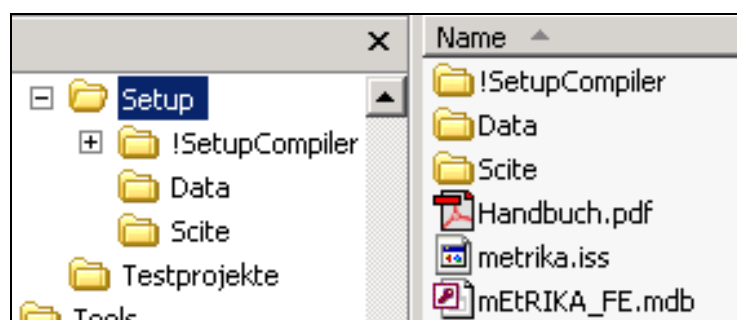


Abbildung 38 – Verzeichnisstruktur zur Generierung eines Installationspakets

6.5 Vergleich der mEtRIKA Ergebnisse mit der LOGISCOPE Validierung

Um die Funktions- und Interpretationsweise der Implementierung darzustellen wird sie folgend an einem Beispiel – Assemblercode aufgezeigt. Testgegenstand ist eine Codesequenz, die aus einem im BIA entwickelten CPU – Test stammt und in diesem Kontext eine Teilkomponente darstellt. Der CPU – Test wurde von Mario Mai entwickelt und umfasst in seiner vollständigen Version mehrere hundert Einzelkomponenten. Die ausgewählte Komponente „ACC_TEST“ dient im Rahmen der Prozessortests zur Prüfung des Akkumulators anhand einer „wandernden“ „1“.

Anhand des Codebeispiels wird folgend verdeutlicht, wie sich die wesentlichsten Basiskennzahlen für die Berechnung der Gesamtmetriken zusammensetzen und aus dem Prüfling herleiten lassen. Darüber hinaus wurde mEtRIKA mit bereits zertifizierten Industrieanwendungen getestet und die Ergebnisse mit denen von Krell [Krell 2003] Ermittelten verglichen.

```

2 ;metrika ACC_TEST
3 ACC_TEST:  CLR  C
4             MOV  R1,#08             ; R1 sei Schleifen„hler
5             MOV  A,#01             ; Testbyte: 0000$0001
6 ACC_1:      RLC  A                 ; wandernde 1 durch Carry-Flag
7             JC   ACC_2             ; Wenn 1 im Carry angekommen Springe zu ACC_2
8             DJNZ R1, ACC_1         ; Decrementiere Schleifen„hler
9 ACC_2:      DEC  R1                 ; Decrementiere Schleifen„hler nochmals
10            CJNE R1, #00h, ACC_ERR  ; hier geschieht der eigentliche Akkutest:
11            SJMP ACC_ENDE          ; Ende der Routine
12 ACC_ERR:   MOV  FEHLER,#FEHLER_ACC ; Fehlerspeicher laden
13            SJMP ACC_ERR           ; Endlosschleife
14 ACC_ENDE:  RET
15

```

Abbildung 39 – Untersuchungsgegenstand „ACC_TEST“ – Quellcode

Zur Analyse des Quellcodes wurden in einem ersten Schritt die Referenzdaten angelegt und konfiguriert. Die mit grün markierten Ausdrücke entsprechen den „Labels“ des Codes. Stehen sie hinter einem Befehl gelten sie als Parameter und wer

den bei der Berechnung der Hasteed – Metriken als Operanden gewertet. Mit dunkelblau wurden die Befehlsausdrücke markiert. Sie unterscheiden sich in den Klassifikationen:

- 0: Instruktion, die nicht zu analysieren ist
- 1: Instruktion, die zu analysieren ist
- 2: Instruktion entspricht einem Prozeduraufruf
- 3: Modulende ungleich einem normalen Ende der Komponente
- 4: Bedingter Sprung („if“)
- 5: Sprung, der nicht von einer Bedingung abhängig ist („goto“)
- 6: Modulende

Nach den Klassifikationen der Ausdrücke variieren die berechneten Ergebnisse. Soll ein Befehl beispielsweise in der Untersuchung nicht betrachtet werden, wird er mit „0“ klassifiziert, was zur Folge hat, dass er selbst bei den Halstead – Metriken nicht als Operator und seine Parameter nicht als Operanden eingehen. Ein Befehl der Klasse „3“ erhöht die Anzahl der „weiteren Ausgänge (P_NODES)“ in den McCabe – Metriken, da er ein nicht „gewöhnliches“ Modulende repräsentiert.

Alle weiteren Codeelemente des Prüflings stellen auskommentierten Text, beziehungsweise Parameter der Befehle dar.

Gemäß den Definitionen aus Kapitel 3.2.5 und [Krell 2003] betragen die Halstead – Metriken bezüglich des Codebeispiels aus Abbildung 39 nach mEtRIKA:

Metrik:	Von mEtRIKA / Krell ermittelt:
• Anzahl der Operatoren (N1):	12 / 21
• Anzahl der Operanden (N2):	17 / 17
• Anzahl der verschiedenen Operatoren (n1):	9 / 10
• Anzahl der verschiedenen Operanden (n2):	12 / 12
• Anzahl der Kommentarzeilen (N_COM):	10 / 10
• Anzahl der Programmzeilen (N_INST):	12 / 11

Die von Krell, beziehungsweise LOGISCOPE, und von mEtRIKA ermittelten Kennzahlen weisen in diesem Beispiel nur wenige Differenzen auf. Lediglich die Metrik der Operatoren weicht im Kontext zu umfangreicheren Komponenten verstärkt ab.

Die Zählweise die LOGISCOPE dabei verwendet ist unklar und kann daher nicht deutlich gegenüber der mEtRIKA – Zählweise abgegrenzt werden.

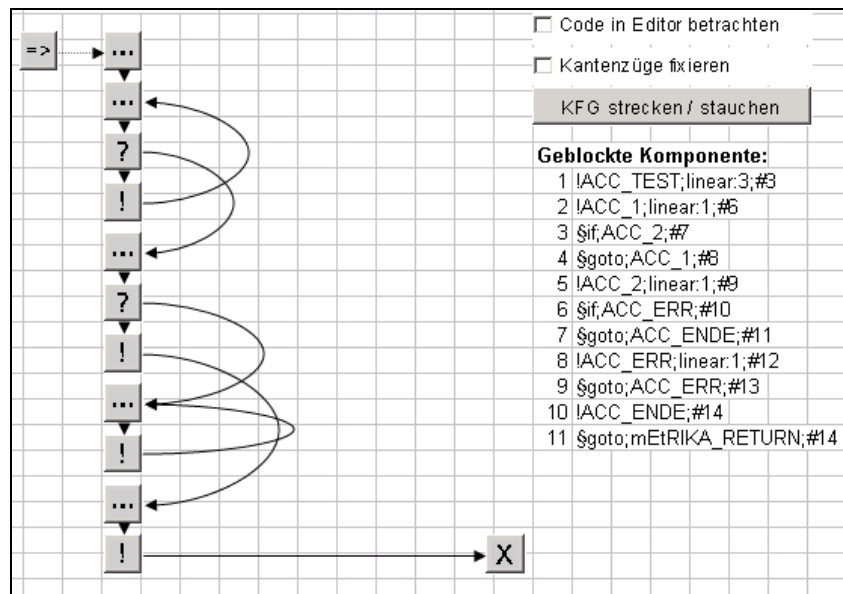


Abbildung 40 – Untersuchungsgegenstand „ACC_TEST“ – KFG

Die Berechnung der McCabe – Metriken gestaltet sich im Vergleich zur Generierung der Halstead – Metriken aufwändiger. Nach Anhang - 1 bilden die zu ermittelnden wesentlichen Grundmetriken bei McCabe die Anzahl der „weiteren Ausgänge“ (P_NODES), die Anzahl der „Eingänge“ (N_IN), die Anzahl der „Knoten“ (N_NODES), die Anzahl der „verarbeitenden Knoten“ (N_SEQ) und die Anzahl der „Sprünge/Verzweigungen“ (N_JUMPS).

Um die Korrektheit der ermittelten Metriken leichter aufzuzeigen, stellt Abbildung 40 den Kontrollflussgraphen und den „geblockten“ Pseudocode des Prüflings dar. Ein Auszug der Einzelmetriken ergibt:

Metrik:	Von mEtRIKA / Krell ermittelt:
• Anzahl der „weiteren Ausgänge“ (P_NODES):	0 / 0
• Anzahl der „Eingänge“ (N_IN):	1 / 1
• Anzahl der „Knoten“ (N_NODES):	11 / 11
• Anzahl der „Kanten“ (N_EDGES):	13 / 13
• Anzahl der „verarbeitenden Knoten“ (N_SEQ):	4 / 4
• Anzahl der „Sprünge/Verzweigungen“ (N_JUMPS):	6 / 5

Als „weitere Ausgänge“ werden Befehle der Klassifikation „3“ und Sprünge, die auf Sprungmarken verweisen die nicht innerhalb der Komponente liegen, interpretiert. „Eingänge“ sind, im Kontext der Applikation mEtRIKA, Moduleinsprünge über ein Modullabel. Die Anzahl der „Knoten“ entspricht Zeilen der geblockten Komponente, da jede Zeile einen Knoten im Kontrollflussgraphen repräsentiert. Um die Anzahl der „verarbeitenden Knoten“ zu ermitteln werden alle Knoten der geblockten Komponente betrachtet, die Instruktionen beinhalten. Besteht ein Knoten lediglich aus einem Label, wie es beim Knoten 10 der Fall ist, wird dieser nicht als „verarbeitender Knoten“ gewertet. Die Metrik bezüglich der Sprünge und Verzweigungen im Kontrollflussgraphen baut ebenfalls auf die Werte der geblockten Komponente auf. Wesentlich sind dabei die Marken der Knoten, die ein „§“ enthalten, da diese bedingte und unbedingte Sprünge darstellen. Ist eine Sprungmarke nicht im Kontext der betrachteten Komponente bekannt, wird dieser Sprung als „undefiniert“ gewertet. Die Kennzahlen zur Anzahl der Kanten und zur zyklomatischen Zahl werden beim Analysevorgang nicht ermittelt sondern aus den Grundmetriken arithmetischen berechnet. Dennoch sind sie Bestandteil der Aufstellung, da sie wichtige Metriken im Zusammenhang zu den Strukturmetriken darstellen.

Beim Vergleich umfangreicherer Komponenten sind Abweichungen insbesondere bei den Metriken Anzahl der Knoten und Anzahl der Sprünge aufgetreten. Bei Berechnung der Knoten ist eine abweichende Art der „Blockung“ in der LOGISCOPE – Implementierung bei Erstellung des Pseudocodes die Ursache. Demnach variieren die Knoten um bis zu vier Einheiten bei den Tests. Die Anzahl der Sprünge ist im Analysevorgang stark von den Befehlsklassifikationen in den Referenzdaten abhängig und variiert in den Testergebnissen mit bis zu zwei Einheiten. Diese Kennzahlen sind Ursache bei Abweichungen der zyklomatischen Zahl, die wesentliches Maß der McCabe – Metriken darstellt.

	ACC_TEST		Komponente01		Komponente02		Komponente03	
	Krell	mEtRIKA	Krell	mEtRIKA	Krell	mEtRIKA	Krell	mEtRIKA
Halstead								
INST	11	12	71	72	26	29	28	38
COMM	10	10	72	72	28	56	13	16
OOPT N1	21	12	178	72	125	29	49	38
OPET n1	10	9	11	10	18	11	8	8
OOPN N2	17	17	142	142	95	104	28	34
OPEN n2	12	12	29	29	32	36	17	20
McCabe								
P_NODES	0	0	1	1	0	0	2	0
N_OUT	1	1	2	2	1	1	3	1
N_IO	2	2	3	3	2	2	4	2
N_EDGES	13	13	77	82	2	2	11	17
N_NODES	11	11	53	57	3	3	12	15
VG	4	4	27	28	1	1	3	4
MAX_DEG	3	3	16	16	2	2	5	4
N_SEQ	4	4	21	22	1	1	5	9
N_JUMPS	5	6	29	31	0	1	3	5
Krell								
N_P_NODES	0	0	1	1	0	0	2	0
S_MAX_DEG	0	0	14	12	0	0	0	0
VG_S_MAX_DEG	4	4	13	16	1	1	3	4
Qualitätskriterien								
Testbarkeit	Akzeptiert	Akzeptiert	Zu kürzen	Zu kürzen	Akzeptiert	Akzeptiert	Zu strukturieren	Akzeptiert
Einfachheit	Akzeptiert	Akzeptiert	Zu testen	Zu inspizieren	Zu dokumentieren	Akzeptiert	Akzeptiert	Zu dokumentieren
Lesbarkeit	Akzeptiert	Akzeptiert	Zu inspizieren	Zu dokumentieren	Akzeptiert	Akzeptiert	Zu dokumentieren	Akzeptiert
Selbstbeschreibung	Akzeptiert	Akzeptiert	Akzeptiert	Akzeptiert	Akzeptiert	Akzeptiert	Akzeptiert	Akzeptiert
Global	Akzeptiert	Zu dokumentieren	Zu testen	Zu inspizieren	Akzeptiert	Zu dokumentieren	Zu dokumentieren	Zu dokumentieren

Abbildung 41 – Übersicht Testergebnisse mEtRIKA

Abstrahiert von den Einzelmetriken werden die Qualitätskriterien zur „ersten“ Beurteilung eines Prüflings verwendet. Bei durchgeführten Testläufen lassen sich jedoch trotz der vereinzelt festgestellten Abweichungen in den Metriken nur wenige Abweichungen in den Qualitätskriterien aufzeigen. In einigen Fällen sind Differenzen um eine Einheit aufgetreten, die eine Komponente beispielsweise bezüglich der „Einfachheit“ nicht in „Zu testen“ sondern in „Zu inspizieren“ eingestuft haben.

7 Zusammenfassung und Ausblick

Die prototypische Realisierung von mEtRIKA im Rahmen dieser Abschlussarbeit zeigt, dass die Anforderungen einer Prüf- und Zertifizierungsstelle an eine Rechnerunterstützung bei der Softwareanalyse weitestgehend in Form einer neuen Applikation umzusetzen waren. Für die Gutachter bedeutet dies, dass sie zukünftig nicht mehr ausschließlich die starren Funktionalitäten der bisherigen Anwendung nutzen müssen, sondern zukünftig verstärkt mEtRIKA in die Softwareprüfung einbeziehen können. Innerhalb des kurzen Entwicklungszeitraums blieb nur wenig Zeit zur ausführlichen Verifikation und Validation der in mEtRIKA implementierten Funktionen, so dass die Anwendung, wie bereits angeschnitten, prototypischen Charakter aufweist und daher jetzt noch nicht bedenkenlos zum Einsatz kommen kann. Jedoch zeigt die Entwicklung ebenfalls, dass mit ihr eine Ideenschnittstelle geschaffen wurde, die Raum für jegliche Anforderungen der Anwender bezüglich Funktionalität, Handhabung, Darstellung und Berichtswesen im Kontext einer Softwareprüfung erlaubt. Die in Kapitel 4.4.3 formulierten Anforderungen konnten durch die mEtRIKA - Realisierung folgendermaßen umgesetzt werden:

- Über die offene Programmierschnittstelle VBA kann mEtRIKA auch in Zukunft in seinem Automatisierungsgrad und bezüglich weiterer Funktionalitäten ausgebaut werden. Die Modularisierung des ersten Entwicklungsentwurfs ermöglicht die komponentenweise Weiterentwicklung, beispielsweise bezüglich bestimmter Analysemodule.
- Durch die Realisierung auf Basis einer Microsoft – Office – Plattform können angebundene Kommunikationsinfrastrukturen vielseitig genutzt werden. Außerdem wurde dadurch der Grundstein zur leichten Kopplung von Büroanwendungen zur Bearbeitung von Softwareprüfungen gelegt.
- Eine intuitiv verstehbare Benutzerführung vereinfacht Anwendern Einstellungen im Kontext einer Analyse zu bearbeiten und verhilft effizient die Applikation bedienen zu können.
- Die Datenhaltung in Form einer zentralen Projektdatei gestattet die leichte Integration von Datensicherungsmechanismen.
- Mit mEtRIKA wird eine weitgehende Rechnerunterstützung zur Konfiguration der Referenz- und Verwaltungsdaten geboten

Über diese realisierten Anforderungen hinaus bestehen zahlreiche weitere, die es zukünftig im Rahmen der Applikation umzusetzen gilt. Eine Reihe der wichtigsten sind:

- Bisher steht mEtRIKA ausschließlich zur Analyse von Assemblerprodukten zur Verfügung. Durch die Implementierung eines C – Analysemoduls könnten zukünftig C – Programme im Rahmen einer Softwareprüfung über die Rechnerunterstützung untersucht werden. Diese Modulerweiterung ist darüber hinaus für jede weitere Programmiersprache umsetzbar und daher nicht nur in Bezug auf C zu betrachten.
- Die Teilautomatisierung der Referenzdatenkonfiguration in mEtRIKA ist ein mächtiges Werkzeug, welches in der LOGISCOPE – Implementierung in keiner Weise enthalten ist. Um die Unterstützung zur Verwaltung der Referenzdaten zu erweitern, könnten Importfunktionen entwickelt werden, die es ermöglichen, die Referenzdaten aus bestehenden Analyseprojekten zu importieren. Ferner könnten eigens prozessorabhängige Bibliotheken mit Referenzdaten entwickelt werden, die zum Beispiel durch das Auslesen eines digitalen Datenblatts generiert werden könnten. Eine dritte Möglichkeit vorgefertigte Referenzdaten zu erhalten, könnte das Auslesen von LOGISCOPE – Konfigurationen aus vergangenen Projekten sein.
- Um die Anwendung der in mEtRIKA implementierten Qualitätskriterien zu erweitern, könnte zukünftig vorgesehen werden, neue beziehungsweise projekteigene Kriterien für eine Softwareprüfung zu definieren.
- Die Darstellung und Visualisierung der linguistischen und strukturellen Metriken spielt die wichtigste Rolle im Kontext einer Softwareuntersuchung. In der bisherigen Form implementiert mEtRIKA lediglich eine Darstellungsweise. In Zukunft sollte die bisherige Funktionalität der Kontrollflussgraphen, sowie weitere, ähnlich der in LOGISCOPE realisierten, Visualisierungen ausgebaut und neu entwickelt werden. Die Freiheit neue Darstellungsformen definieren und umsetzen zu können, spiegelt den, in anbetracht der Wichtigkeit von Darstellungen, größten Vorteil der Eigenentwicklung wider. Dabei könnten Büroanwendungen, andere Werkzeuge, oder vollständige Eigenentwicklungen Grundlage sein.

Ferner könnte in diesem Zusammenhang der Ausbau des Berichtswesens anhand Microsoft Excel relevant sein. Dabei könnte der hohe Verbreitungsgrad der Office – Produkte genutzt werden, um die Berichtsmappen via Kommunikationsinfrastrukturen zu verteilen. Über den Inno Setupcompiler können die Berichte samt dem SciTE – Editor und den analysierten Quelltexten zu einem Installationspaket verbunden werden, um es in dieser Form anderen, in die

Prüfung involvierten Instanzen bereitzustellen. Diese hätten dann die Möglichkeit das Paket zu installieren und somit über den SciTE – Editor und die Interaktion mittels VBA auf die Quelltexte und Berichte in graphisch aufbereiteter Form zuzugreifen.

- Ebenso in Bezug auf das Berichtswesen können eine Reihe weiterer Funktionalitäten angedacht werden. Beispiele hierfür sind die Berichterstellung über die VBA Programmierschnittelle von Microsoft Word oder die Berichtsfunktionalitäten, die bereits über Microsoft Access zur Verfügung stehen, jedoch in der bisherigen Implementierung nicht verwendet werden. Denkbar wäre auch die Ansteuerung eines PDF – Druckertreibers, der die Berichte in das weit verbreitete PDF – Dateiformat transformiert.

Die obigen Ausführungen zu zukünftig zu implementierender Funktionalitäten zeigen das Potential, das durch die erstmalige Realisierung des mEtRIKA – Prototypen entstanden ist. Fortführend sind eine Vielzahl weiterer Anpassungen des Frontend und dessen interner Verarbeitung denkbar. Diese könnten sich von der Protokollierung der Analysevorgänge, über Konsistenzprüfungen beim Öffnen der Projektdatei bis hin zur Einbindung eines Menüs mit den wichtigsten Funktionen zur komfortableren Nutzung des Frontend für den Anwender erstrecken.

Alles in allem zeigt die Implementierung der Applikation mEtRIKA, dass weitreichende Funktionalitäten nach zukünftigen Benutzeranforderungen programmiert werden können, um es als effizientes und vielseitiges Werkzeug bei Softwareprüfungen einzusetzen.

Literaturverzeichnis

- [Balzert 2000] Balzert, H.: Lehrbuch der Software – Technik: Software – Entwicklung. 2. Auflage. Spektrum, Akademischer Verlag, Berlin, Heidelberg 2000
- [Balzert 1998] Balzert, H.: Lehrbuch der Software – Technik: Software – Management, Software – Qualitätssicherung, Unternehmensmodellierung. Spektrum, Akademischer Verlag, Berlin, Heidelberg 1998
- [Bömer et al 1998] Bömer, T.; Bülesbach, K.-H.; Gnedina, A.; Grigulewitsch, W.; Reinert, D.; Reuß, G.; Schaefer, M.: Programmierregeln für die Erstellung von Software für Steuerungen mit Sicherheitsaufgaben. Wirtschaftsverlag NM – Verlag für neue Wissenschaft GmbH, Berlin, Dortmund 1998
- [Born 1999] Born, G.: Microsoft Office 2000 Programmierung: Grundlagen, Beispiele, Lösungen. Microsoft Press Deutschland, Unterschleißheim 1999
- [Consolini 2001] Consolini, L.: Perspectives: Introduction to the Subject Domain. / Software Quality Approaches: Testing, Verification and Validation – Software Best Practise 1, S. 33. Springer-Verlag, Berlin, Heidelberg 2001
- [DGQ 1992] Deutsche Gesellschaft für Qualität e.V. Ffm. (Hrsg.): Methoden und Verfahren der Software-Qualitätssicherung. Beuth-Verlag GmbH, Berlin, 1992
- [Dumke 1992] Dumke, R.: Softwareentwicklung nach Maß: Schätzen – Messen – Bewerten. Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig, Wiesbaden 1992

-
- [Dumke, Winkler 1999] Dumke, R.; Winkler, A.: Y2K from a Metrics Point of View. – Software Measurement: Current Trends in Research and Practice, S. 173. Deutscher Universitätsverlag, Gabler Edition Wissenschaft, Wiesbaden 1999
- [Ehrenberger 2002] Ehrenberger, W.: Software-Verifikation – Verfahren für den Zuverlässigkeitsnachweis von Software. Carl HanserVerlag, München, Wien 2002
- [Edwards et al 1999] Edwards, S.; Henry, S.; Bodnar, R.: Software Metrics for Multimedia Languages. – Software Measurement: Current Trends in Research and Practice, S. 195. Deutscher Universitätsverlag, Gabler Edition Wissenschaft, Wiesbaden 1999
- [Fenton, Pfleeger 1997] Fenton, N. E.; Pfleeger, S. L.: Software Metrics – A Rigorous and Practical Approach. Second Edition. PWS Publishing Company, Boston 1997
- [Gilb 1977] Gilb, T.: Software Metrics. Winthrop Publishers, Inc.; Cambridge, Massachusetts 1977
- [Halstead 1977] Halstead, M. H.: Elements of Software Science. Elsevier North-Holland, Inc.; New York 1977
- [HVBG 2004] Hauptverband Berufsgenossenschaften (Hrsg.): Das Berufsgenossenschaftliche Institut für Arbeitsschutz (BIA). <http://www.hvbg.de/d/bia/index.html> (08.06.2004)
- [Jacobsen-Rey 2000] Jacobsen-Rey, M.: Automated Software Inspection – Attaining New Levels of Software Quality. / Software – Metriken: Entwicklungen, Werkzeuge und Anwendungsverfahren, S. 1. Deutscher Universitätsverlag, Gabler Edition Wissenschaft, Wiesbaden 2000

- [Klug, Schaefer 1997] Klug, J.; Schaefer, M.: Fehlererkennende Maßnahmen in Mikroprozessoren / Sicherheitstechnisches Informations- und Arbeitsblatt. Berufsgenossenschaftliches Institut für Arbeitsschutz (BIA) – Handbuch 29. Lfg. VII/97, Sankt Augustin 1997
- [Krell 2003] Krell, M.: Bestimmung von Qualitätskriterien für sicherheitsrelevante Software im Maschinenschutz auf Basis von zertifizierten Industrieanwendungen. Diplomabschlussarbeit - Fachhochschule Bonn-Rhein-Sieg / Fachbereich Informatik, Sankt Augustin 2003
- [Lewerentz et al 2000] Lewerentz, C.; Rust, H.; Simon, F.: Quality – Metrics – Numbers – Consequences. / Software – Metriken: Entwicklungen, Werkzeuge und Anwendungsverfahren, S. 51. Deutscher Universitätsverlag, Gabler Edition Wissenschaft, Wiesbaden 2000
- [Orci 2001] Orci, T.: Software Metrics Applications in a European Perspective. / Software Process Improvement: Metrics, Measurement and Process Modelling – Software Best Practice 4, S. 71. Springer-Verlag, Berlin, Heidelberg 2001
- [Paradiso 2001] Paradiso, M.: Software Verification & Validation Introduced. / Software Quality Approaches: Testing, Verification and Validation – Software Best Practise 1, S. 36. Springer-Verlag, Berlin, Heidelberg 2001
- [Reinert 2001] Reinert, D.: Sicherheitstechnische Grundlagen / Sichere Bussysteme für die Automation, S. 12. Hüthig GmbH & Co. KG, Heidelberg 2001
- [Reinert, Reuß 1991] Reinert, D.; Reuß, G: Sicherheitstechnische Beurteilung und Prüfung mikroprozessorgesteuerter Schutzeinrichtungen / Sicherheitstechnisches Informations- und Arbeitsblatt. Berufs-

- genossenschaftliches Institut für Arbeitsschutz (BIA) – Handbuch 17. Lfg. X/91, Sankt Augustin 1991
- [Sommerville 2003] Sommerville, I.: Software Engineering. 6. Auflage. Pearson Studium, München 2003
- [VERILOG 1993a] VERILOG GmbH (Ed.): LOGISCOPE Assembler Analyzers – Reference Manual. München 1993
- [VERILOG 1993b] VERILOG GmbH (Ed.): LOGISCOPE Editor Graphical User Interface 1.1 – Reference Manual. München 1993
- [VERILOG 1993c] VERILOG GmbH (Ed.): LOGISCOPE Viewer – Reference Manual. München 1993
- [Vogel-Heuser 2003] Vogel-Heuser, B.: Systems Software Engineering. Oldenbourg Industrieverlag GmbH, München 2003
- [Windpassinger 2000] Windpassinger, H.: Möglichkeiten der Metrik – basierten Modellierung und Auswertung von Qualitätsvorgaben mit dem Werkzeug LOGISCOPE. / Software – Metriken: Entwicklungen, Werkzeuge und Anwendungsverfahren, S. 135. Deutscher Universitätsverlag, Gabler Edition Wissenschaft, Wiesbaden 2000
- [Zuse 1999] Zuse, H.: Thirty Years of Software Measurement. – Software Measurement: Current Trends in Research and Practice, S. 3. Deutscher Universitätsverlag, Gabler Edition Wissenschaft, Wiesbaden 1999

Anhang - 1 In mEtRIKA implementierte Metriken – Übersicht

Folgende Tabellen erläutern, welche Metriken durch die Anwendung mEtRIKA umgesetzt und bei einer Softwareanalyse berechnet werden. Handelt es sich um Kennzahlen, die direkt aufgrund der zu untersuchenden Software ermittelt werden, steht in der Spalte Berechnung „Ermittelt“ sowie ein Zusatz, innerhalb welcher Funktion dies geschieht. Bei Metriken, die sich wiederum aus anderen zusammensetzen, enthält die rechte Spalte die Formel, mit der die Metrik berechnet wird. Die Kurzbezeichnungen sind Abkürzungen, wie sie für die Metriken in der LOGISCOPE – Implementierung beziehungsweise in den Definitionen von Krell [Krell 2003] verwendet werden.

Bezeichnung der Metrik	Kurzbez.	Berechnung
------------------------	----------	------------

Metriken von Halstead:

Anzahl Operatoren	N1	Ermittelt (metHalstead)
Anzahl Operanden	N2	Ermittelt (metHalstead)
Anzahl verschiedene Operatoren	n1	Ermittelt (metHalstead)
Anzahl verschiedene Operanden	n2	Ermittelt (metHalstead)
Anzahl Kommentarzeilen	N_COM	Ermittelt (metHalstead)
Anzahl Programmzeilen	INST	Ermittelt (metHalstead)
Frequenz Kommentar	COM_R	$N\_COM / INST$
Programmlänge	PR_LGHT	$N1 + N2$
Anzahl Vokabeln	VOC_SZ	$n1 + n2$
Frequenz Vokabular	VOC_F	PR_LGHT / VOC_SZ
Größe der Instruktionen	AVG_S	$PR\_LGHT / INST$
Programmvolumen	V	$PR\_LGHT * \text{Log} (VOC\_SZ) / \text{Log} (2)$
Potentiell Volumen	VS	$(N2 + 2) * \text{Log} (N2 + 2) / \text{Log} (2)$
Programm – Level	PR_LVL	VS / V
Schätzwert für Programmlevel	DL	$2 / n1 * n2 / N2$
Information Komponente	INTELL	$V * DL$
Programmkomplexität	D	$1 / PR\_LVL$
Aufwand	EFFORT	V / PR_LVL

Fehlerwahrscheinlichkeit	N_ERRORS	EFFORT ^{2/3} / 3000
--------------------------	----------	------------------------------

Metriken von McCabe:

Anzahl weiterer Ausgänge	P_NODES	Ermittelt (metMcCabe)
Anzahl Ausgänge	N_OUT	P_NODES + 1
Anzahl Eingänge	N_IN	Ermittelt (metBerechnenGlobaleMetriken)
Anzahl Ein- und Ausgänge	N_IO	N_OUT + N_IN
Anzahl Knoten	N_NODES	Ermittelt (metMcCabe)
Anzahl Kanten	N_EDGES	N_NODES – 1 + N_JUMPS – UNCOND_JUMPS
Zyklomatische Zahl	VG	N_EDGES – N_KNOTEN + N_IO
Kontrollflussdichte	C_DENS	(VG - 1) / N_NODES
Anzahl nicht zyklischer Wege	N_PATHS	Ermittelt (metBerechnenGlobaleMetriken)
Maximaler Grad	MAX_DEG	Ermittelt (metMcCabe)
Anzahl verarbeitende Knoten	N_SEQ	Ermittelt (metMcCabe)
Anzahl Sprünge / Verzweigungen	N_JUMPS	Ermittelt (metMcCabe)
Anzahl undefinierte Sprünge	UNDEF_JUMPS	Ermittelt (metMcCabe)
Anzahl unbedingte Sprünge	UNCOND_JUMPS	Ermittelt (metMcCabe)
Anzahl direkt aufgerufene Komponenten	DRCT_CALLS	Ermittelt (metMcCabe)
Wesentliche Komplexität	ESS_CPX	Geblockt(N_EDGES) - Geblockt(N_NODES) + N_IO

Metriken von Krell:

Durchschnittlich verwendete Operatoren	N1_F	N1 / n1
Durchschnittlich verwendete Operanden	N2_F	N2 / n2
Anzahl unterschiedlicher Aussprünge	NP_NODES	Ermittelt (metMcCabe)
Anzahl indirekte Aussprünge	IP_NODES	Ermittelt (metKrell)

Summe aller maximalen Grade	S_MAX_DEG	Ermittelt (metMcCabe)
Anzahl „tote“ Sprungmarken	N_DEAD_LABELS	Ermittelt (metKrell)
Anzahl überflüssiger Sprungmarken	N_WASTE_JUMPS	Ermittelt (metKrell)
Anzahl einzeliger Schleifen	N_LOOP_LINE	Ermittelt (metKrell)
Anzahl Schleifen	N_LOOPS	Ermittelt (metKrell)
VG ohne Case – Strukturen	VG_S_MAX_DEG	VG – S_MAX_DEG
VG ohne Fehlerverwaltung	VG_NEU	VG – S_MAX_DEG – IP_NODES – NP_NODES
VG * Anzahl Programmzeilen	VG_STMTS	VG * INST
Erweiterte Frequenz – Kommentare	NEU_COM_R	$N\_COM * VOC\_F / (INST + (VG\_S\_MAX\_DEG - 1) * 2)$

Anhang - 2 Handbuch – mEtRIKA

mEtRIKA

Analyse sicherheitsrelevanter Software Handbuch

Dokument: Handbuch.doc
Autor: Christian Staron
Datum: 27. Juli 2004

Inhaltsverzeichnis

Inhaltsverzeichnis	2
Abbildungsverzeichnis	3
1 Die Anwendung mEtRIKA	4
1.1 Aufbau und Architektur der Applikation	5
1.2 Einbindung weiterer Applikationen	7
1.2.1 SciTE	7
1.2.2 Microsoft Excel 2002	7
1.3 Funktionsumfang	7
1.3.1 Projektverwaltung	8
1.3.2 Einstellen Projektdaten	8
1.3.3 Einstellungen Sprache / Dialekt	8
1.3.4 Kennzahlen	10
1.3.5 Darstellung	10
2 Bedienung über das Graphical User Interface (GUI) am Beispiel einer Softwareprüfung	11
2.1 Installation des Frontend „mEtRIKA“	11
2.2 Start der Anwendung	12
2.3 Erste Schritte in der Projektverwaltung	15
2.4 Verwaltung und Konfiguration der Projektdaten	19
2.5 Einstellungen zu Programmiersprache / Dialekt der zu untersuchenden Software und Start der Berechnung	29
2.6 Berechnete Metriken und weitere Kennzahlen	36
2.7 Darstellungsfunktionen	39

Abbildungsverzeichnis

Abbildung 1 – Architektur mEtRIKA	5
Abbildung 2 – Hauptformular mEtRIKA	6
Abbildung 3 – Einordnung der Metriken in das Qualitätsmodell	9
Abbildung 4 – Setup mEtRIKA	11
Abbildung 5 – Legende Ablaufdiagramm einer Softwareprüfung	13
Abbildung 6 – Ablauf einer Softwareprüfung	15
Abbildung 7 – Dialog Projekt anlegen	16
Abbildung 8 – Speichern der Projektdatei	16
Abbildung 9 – Ampelsystem und aktualisieren des Projektpfads	18
Abbildung 10 – Dateidialog zur Aktualisierung des Projektpfads	19
Abbildung 11 – Formular Projekt- & Kundendaten	20
Abbildung 12 – Formular projektspezifische Einstellungen	22
Abbildung 13 – Formular Quelldateien	24
Abbildung 14 – Quelldateien Ampelsystem	26
Abbildung 15 – Dialog „nicht existierende“ Quelldateien neu verknüpfen	27
Abbildung 16 – Quelldateien ohne Textmarken	28
Abbildung 17 – Dialog Quelldateien „mit Textmarken versehen“	28
Abbildung 18 – Formular Einstellungen Programmiersprache – Referenzdaten einlesen	30
Abbildung 19 – SciTE – Editor nicht "ge-high-light-et" & "ge-high-light-et"	31
Abbildung 20 – Formular Einstellungen Programmiersprache manuell bearbeiten	32
Abbildung 21 – Formular Einstellungen Qualitätskriterien	34
Abbildung 22 – Formular Einstellungen Programmiersprache – Metriken erstellen	35
Abbildung 23 – Windows – Taskmanager: Prozesspriorität herabsetzen	36
Abbildung 24 – Formular Kennzahlen	37
Abbildung 25 – Formular Qualitätskriterien	39
Abbildung 26 – Formular Darstellung	40
Abbildung 27 – Beispiel eines Excelberichts – Deckblatt	41
Abbildung 28 – Beispiel eines Excelberichts – Kontrollflussgraph einer Komponente	42
Abbildung 29 – Beispielfunktionen durch Verwendung von VBA im Berichtswesen	44
Abbildung 30 – Beispielwerkzeug Steuerelement – Toolbox, Markierung, Notiz	45

1 Die Anwendung mEtRIKA

„mEtRIKA“ ist eine unter „Visual Basic for Applications“ (VBA) entwickelte Software, die zur Erstellung von Metriken, Qualitätskriterien und Strukturgraphen sicherheitsrelevanter Software dient. Ziel der Anwendung ist, den Gutachtern solcher kritischer Anwendungen einen schnellen Einstieg in die Prüflinge zu ermöglichen. Berechnete Daten dienen als Beurteilungs- und Diskussionsgrundlage für Softwarezertifizierungen und Reviews, in denen die zu prüfenden Anwendungen untersucht werden.

Als Laufzeitumgebung verwendet mEtRIKA Microsoft Access 2002, da es unter dieser Entwicklungsumgebung in VBA implementiert wurde und auf die damit verbundenen Bibliotheken zugreift.

Im Rahmen einer Bachelor – Abschlussarbeit wurde mEtRIKA 2004 in der ersten Version als Prototyp für zu untersuchende Assemblersoftware realisiert, um die bisher eingesetzte Applikation LOGISCOPE der Firma VERILOG als Analysewerkzeug im Softwarezertifizierungsprozess zukünftig abzulösen. Weite Teile dieser Dokumentation wurden aus der Bachelor – Thesis: [Staron, C.: Entwicklung eines Analysewerkzeugs zur Ermittlung von Metriken und Qualitätskriterien sicherheitsrelevanter Software im Maschinenschutz. Bachelor – Abschlussarbeit - Fachhochschule Bonn-Rhein-Sieg / Fachbereich Informatik, Sankt Augustin 2004] entnommen und werden dort weiterführend beschrieben. In diesem Handbuch eingesetzte Abbildungen bilden Teile der Bachelor – Thesis und sind daher auch stets in diesem Zusammenhang zu betrachten. Weiterführende Erläuterungen werden daher nicht an dieser Stelle vorgenommen, sondern diesbezüglich auf die Inhalte der Arbeit verwiesen.

Da es sich wie beschrieben um einen Prototypen handelt, ist der Funktionsumfang begrenzt und in der bisherigen Version für zu prüfende Assemblerprogramme weitestgehend nutzbar. In mEtRIKA umgesetzte Analyse – Kennzahlen sind die ebenfalls in LOGISCOPE implementierten Halstead und McCabe – Metriken, sowie die im Rahmen einer Diplomarbeit 2003 entwickelten Metriken von Krell. Letztere setzen teils auf die Kennzahlen von Halstead und McCabe auf, verknüpfen diese zu hybriden Metriken, oder wurden vollständig neu entwickelt.

1.1 Aufbau und Architektur der Applikation

Die Softwarearchitektur beschreibt sich über zwei Hauptbestandteile und entspricht damit einer zweischichtigen Client – Server – Struktur. Das Herzstück bildet das Frontend, welches als „Fat“ – Client auf dem lokalen Arbeitsplatz des Anwenders, ergo einem Gutachter, installiert wird. „Fat“, da diese Datei die vollständige Verarbeitungslogik enthält. Außerdem sind in ihm alle benötigten graphischen Formulare zur Repräsentation des „Graphical User Interface“ (GUI) eingebunden. Der zweite Teil der Anwendung besteht aus dem Backend, welches ebenfalls eine Microsoft Access – Datei („.mdb“) ist und alle nötigen Tabellen mit Datensätzen enthält. Ein Backend kann durch das Frontend erzeugt werden und dient der Datenhaltung aller, während der Projektbearbeitung anfallenden Daten. Wird eine Projektdatei erzeugt, ist diese mit dem Frontend zu öffnen um die Bearbeitung der Daten zu ermöglichen.

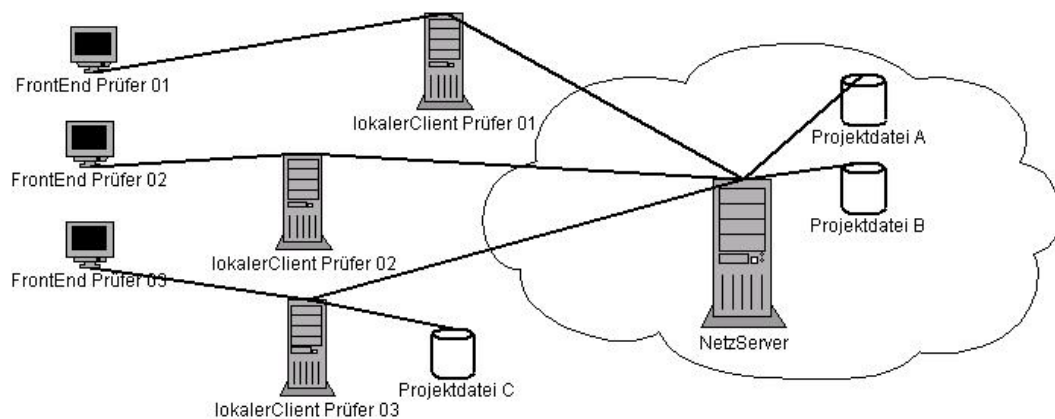


Abbildung 1 – Architektur mEtRIKA

Wie in Abbildung 1 dargestellt, bedient sich mEtRIKA einer verteilten Architektur, wobei die Datenhaltung auf dem, vom Benutzer verwendeten lokalen Client selbst, oder auf einem Rechner im Netzwerk geschehen kann.

Die Anwendung wurde gemäß dem Prinzip eines „Wizards“ entwickelt, welches die intuitive Handhabung der Software unterstützen soll. Sie ist aus einem Hauptformular aufgebaut, in welches alle weiteren Formulare zur Laufzeit geladen werden, so dass sie als Unterformular sichtbar und zugreifbar werden.

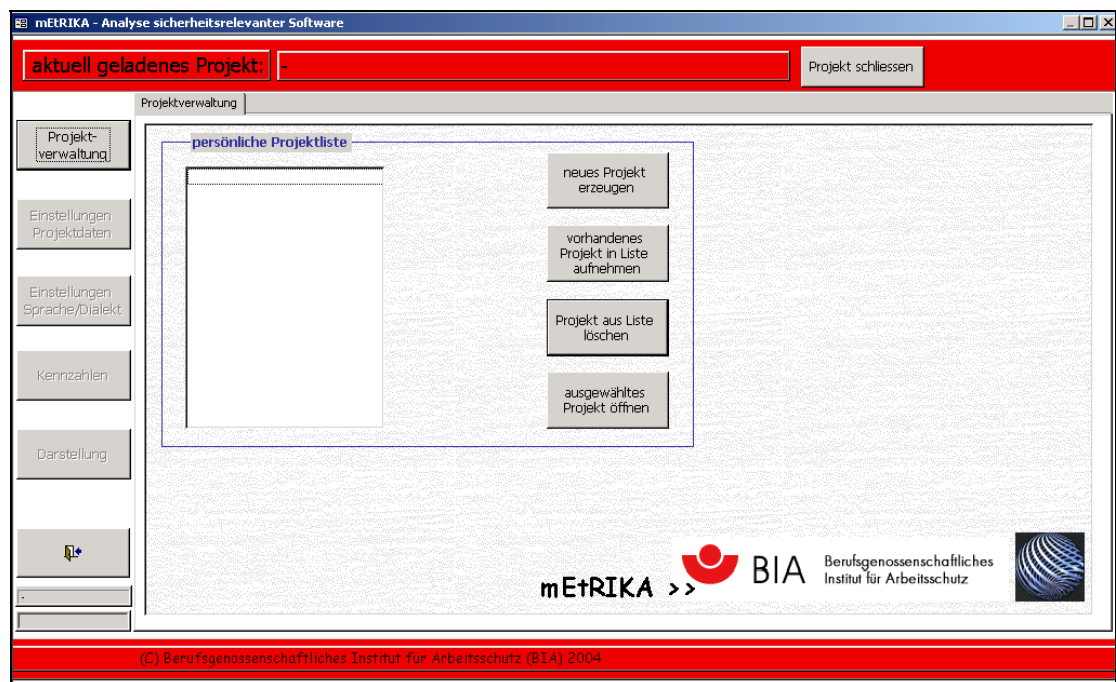


Abbildung 2 – Hauptformular mEtRIKA

Der Kopf des Hauptformulars zeigt an, welches Projekt gerade mit dem Frontend verbunden ist und bearbeitet werden kann. Ist, wie in dem obigen Beispiel, kein Projekt geladen, können die Funktionen der auf der linken Seite befindlichen „Buttons“ nicht verwendet werden.

1.2 Einbindung weiterer Applikationen

Um die Funktionalitäten von mEtRIKA zu erweitern, wurden in das mEtRIKA – Konzept weitere Anwendungen integriert, die über das Frontend automatisiert verwendet werden.

1.2.1 SciTE

Über den Freeware – Editor SciTE lassen sich zu analysierende Codeelemente an vielfachen Stellen der Anwendung betrachten. Durch die Open – Source Entwicklung ist SciTE vielseitig konfigurierbar und daher äußerst flexibel einsetzbar. Globale Einstellungen zum Aufrufverhalten und viele weitere Funktionalitäten können über die Konfigurationsdatei „SciTEGlobal.properties“ im Verzeichnis des Editors genutzt werden. Mit der Installation des Frontend wird SciTE automatisch auf dem Zielrechner im Programmverzeichnis des Analysewerkzeugs installiert.

1.2.2 Microsoft Excel 2002

Die graphische Darstellung der Analyseergebnisse wird durch Microsoft Excel 2002 ermöglicht. Um diese Funktionen nutzen zu können, ist zwingend erforderlich, die XP – Version neben Microsoft Access 2002 installiert zu haben. Ist lediglich eine ältere Excelversion installiert, kann nicht auf die, zur Darstellung benötigten Programmbibliotheken zugegriffen werden, so dass die Generierung von Darstellungen an dieser Stelle nicht möglich ist.

1.3 Funktionsumfang

Die Anwendung ist grundlegend in fünf Funktionsbereiche, gemäß den Buttons auf der linken Seite (Abbildung 2), eingeteilt:

- Projektverwaltung
- Einstellungen Projektdaten
- Einstellungen Sprache / Dialekt
- Kennzahlen
- Darstellung

Mit jedem „Klick“ auf eine dieser Schaltflächen wird das dazugehörige Unterformular im Hauptformular sichtbar und zugänglich. In Abbildung 2 ist gerade die „Projektverwaltung“ sichtbar.

Um eine kurze Übersicht über die Funktionalitäten von mEtRIKA zu vermitteln, werden folgend die wichtigsten Eigenschaften und Mechanismen aufgezeigt. Weiter

führende und detaillierte Zusammenhänge und Abläufe werden unter Punkt zwei erläutert und mit Abbildungen zur Applikation und Markierungen illustriert.

1.3.1 Projektverwaltung

Um im ersten Schritt eine Softwareprüfung durchführen zu können, muss eine Projektdatei, in der die Projektdaten abgelegt werden können, angelegt werden. Wurde diese erzeugt, wird sie automatisch mit der persönlichen Projektliste verknüpft, so dass der Projektname in der Liste sichtbar wird. In dieser Liste können alle, von einem Gutachter bearbeiteten Projekte verwaltet werden und daraus wiederum in das Frontend geladen werden. Diese persönliche Projektliste kann bei jedem Benutzer verschiedene Einträge enthalten, so dass er seine Projekte jederzeit griffbereit hält.

1.3.2 Einstellen Projektdaten

Über die Projektdaten lassen sich alle Informationen bezüglich eines Projekts verwalten. Allgemeine Daten betreffen das Projekt selbst, den Kunden und den Gutachter beim BIA. Darüber hinaus existieren weitere Daten, die projektspezifisch sind, da sie davon abhängen, ob es sich bei der zu prüfenden Software um ein Assemblerprodukt oder ein mit einer anderen Sprache verfasstes Fabrikat handelt. An dieser Stelle sind ebenso Einstellungen für die weitere Verarbeitung, der Berechnung der Metriken, etc. zu leisten. Jede Softwareanalyse setzt auf die zu untersuchenden Quelltexte der Anwendung auf. Einstellungen dazu werden ebenfalls an dieser Stelle getätigt und umfassen beispielsweise die Zuordnung der zu analysierenden Quelltextdateien und deren Einstellungen zur Softwareuntersuchung.

1.3.3 Einstellungen Sprache / Dialekt

Da es sich bei den zu untersuchenden Produkten um Erzeugnisse aus verschiedenen Programmiersprachen handeln kann, müssen Einstellungen zur verwendeten Programmiersprache an dieser Stelle getätigt werden. Ein kurzes Beispiel verdeutlicht die Problematik. Handelt es sich bei dem Prüfling um ein in Assembler entwickeltes Produkt, zeichnet es einen Sprachdialekt aus, der lediglich auf der dafür vorgesehen Mikrocontrollerart ablauffähig ist. Um jedoch sicherheitsrelevante Software für verschiedene Mikrocontroller untersuchen zu können, müssen die Befehlssätze derer, die in der Software verwendet werden, dem Analysemodul zur Verfügung gestellt werden. Die Funktionen zur Einstellung der Sprache / des Dia

lekts gewährleisten dies, indem sie die zu prüfende Software ein erstes Mal analysieren und die enthaltenen Befehle in Listen dem Benutzer zur Verfügung stellen. Der Anwender hat dann die Möglichkeit die einzelnen Schlüsselwörter auf ihre korrekte Klassifizierung hin zu überprüfen. Konfigurationen, wie beispielsweise, „Befehl“ oder „Label“, „bedingter“ oder „unbedingter Sprung“ können fortführend verwaltet werden. Als letzte Funktion wird unter dieser Rubrik die Einstellung von Grenzwerten für Qualitätskriterien ermöglicht. Da mEtRIKA die Analyse von Softwarekandidaten nach den Metriken von Halstead, Krell und McCabe unterstützt, können an dieser Stelle Grenzwerte für die Einzelmetriken spezifiziert werden, die zur Berechnung der übergeordneten Qualitätskriterien nach dem Qualitätsmodell relevant sind.

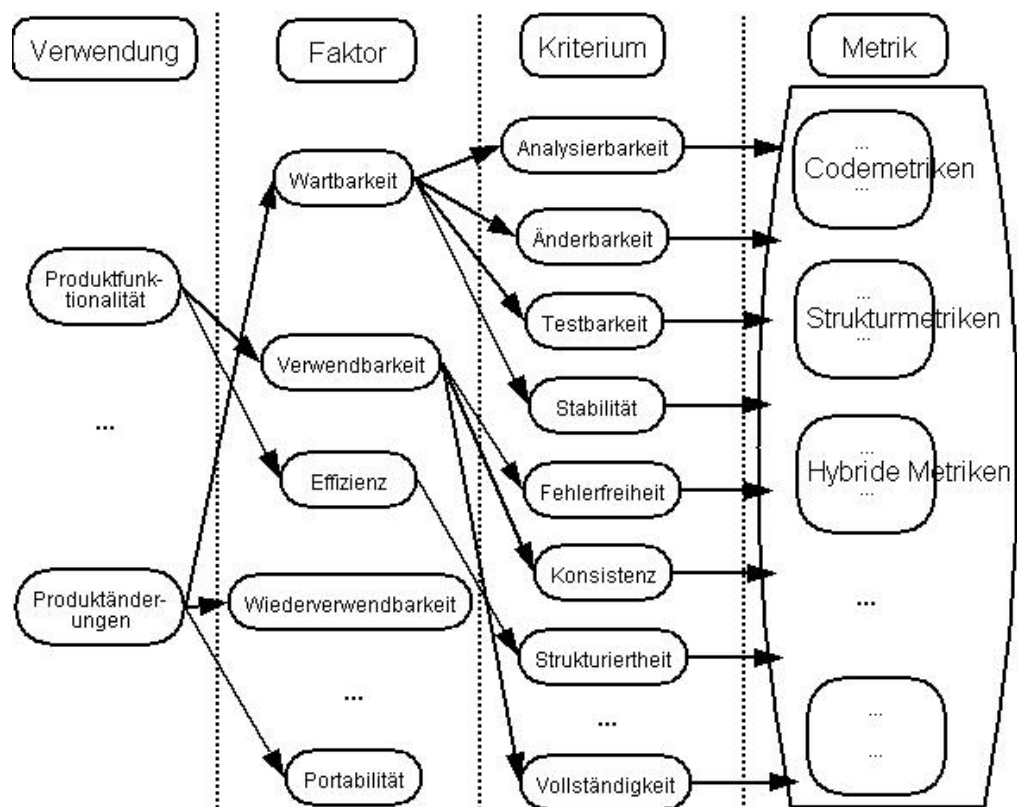


Abbildung 3 – Einordnung der Metriken in das Qualitätsmodell

Das in Abbildung 3 dargestellte Modell bildet das Qualitätsmodell ab, nachdem Softwareeigenschaften durch Faktoren spezifiziert werden. Um diese Faktoren wiederum quantifizieren zu können, werden sie in die Bestandteile einzelner Kriterien aufgeschlüsselt, welche sich wiederum aus einzelnen Codemetriken zusammensetzen lassen.

1.3.4 Kennzahlen

Wurden alle Einstellungen zu den Spracheigenschaften vollständig erledigt, können die Metriken berechnet werden. Die Ergebnisse dieser Berechnungen werden unter dem Formular der Kennzahlen ersichtlich und nutzbar. Eine Liste der Komponenten, die die zu prüfende Software aufweist, unterstützt die Navigation in dem Formular. Der Kopf des Unterformulars zeigt, welche Komponente gerade ausgewählt wurde und welche Metriken dazu angezeigt werden. Darüber hinaus lassen sich unter dem Reiter Qualitätskriterien die globalen Werte zu einer Komponente anzeigen.

1.3.5 Darstellung

Um die Metriken nicht ausschließlich in Zahlenkolonnen auswerten zu müssen, lassen sie sich in Graphiken visualisieren. Unter der letzten Rubrik Darstellung können erfolgreich generierte Metriken in Microsoft Excel 2002 Blättern als Kontrollflussgraphen dargestellt werden.

2 Bedienung über das Graphical User Interface (GUI) am Beispiel einer Softwareprüfung

Um die Nutzung der Anwendung zu erläutern, wird Schritt für Schritt die Applikation an einem Beispiel einer Softwareprüfung aufgezeigt. Dabei zu berücksichtigende Verfahrensschritte werden anhand von Abbildungen und entsprechenden Markierungen auf diesen dargestellt.

2.1 Installation des Frontend „mEtRIKA“

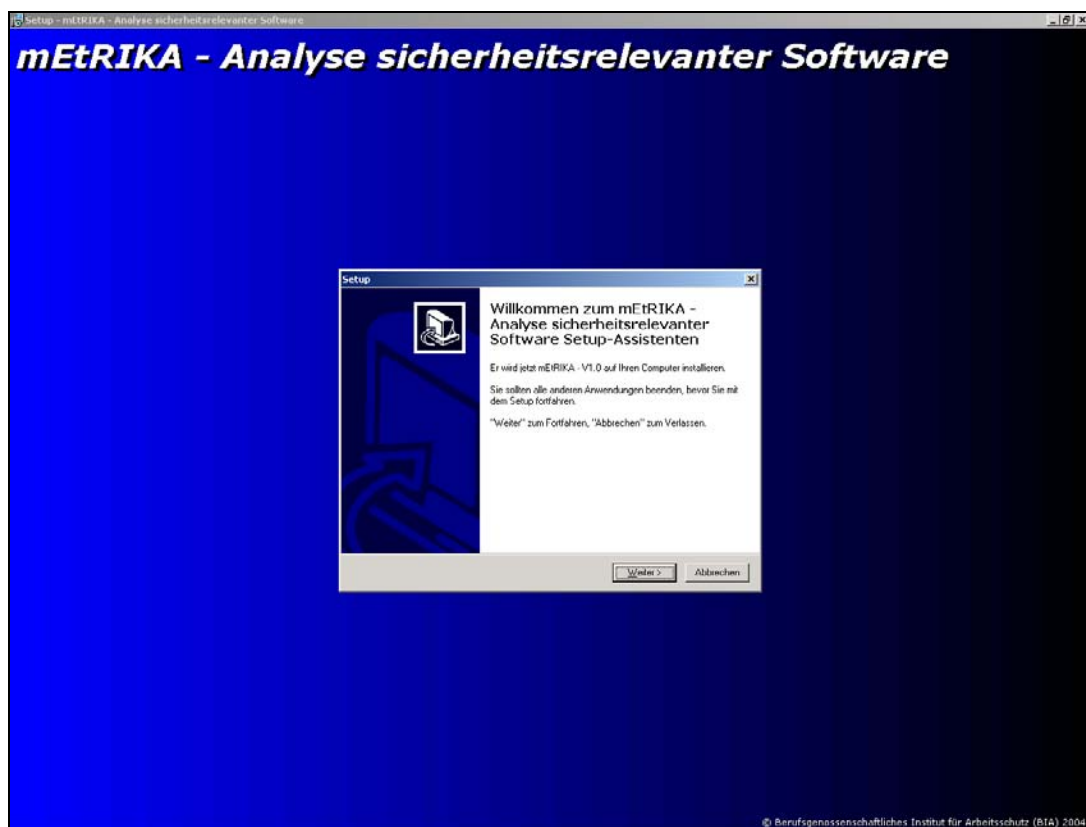


Abbildung 4 – Setup mEtRIKA

Gemäß einem standardisierten Installationsassistenten, werden alle für mEtRIKA erforderlichen Komponenten auf dem lokalen Arbeitsplatz des Anwenders installiert. Bei der Installation unter dem Standardpfad „C:\Programme\mEtRIKA\“ werden die Verzeichnisse „Data“ und „Scite“, sowie die Dateien „mEtRIKA.mdb“, „unins000.exe“ und „unins000.dat“ angelegt. Die „unins*“ – Dateien beinhalten einen Deinstallations – Assistenten, über den die Applikation vom Rechner entfernt werden kann. Die Datei „mEtRIKA.mdb“ entspricht dem Frontend, so dass die Anwendung über diese

Datei gestartet werden kann. Das Verzeichnis „Data“ enthält eine Reihe von Symboldateien, eine Excel – Programmdatei, die bei der Berechnung der graphischen Darstellungen benötigt wird und eine Beispiel – Codedatei mit einem Assemblercode, der zur Beispielhaften Verwendung der Applikation benutzt werden kann.

2.2 Start der Anwendung

Nach der Installation mit den Standardeinstellungen des Installationsassistenten, steht die Anwendung unter „Startmenü“ => „Programme“ => „mEtRIKA“ => „mEtRIKA“ zur Verfügung. Mit Ausführen dieser Verknüpfung wird das Analysewerkzeug gestartet. Über die weiteren Startmenüeinträge lässt sich die Anwendung deinstallieren, beziehungsweise der SciTE – Editor verwenden. Über weitere Links im SciTE – Verzeichnis können Informationen zu dem Editor abgerufen und Einstellungen des Editor vorgenommen werden.

Der vollständige Ablauf einer Softwareprüfung anhand des Analysewerkzeugs mEtRIKA wird in Abbildung 6 dargestellt. Um die Einzelschritte zu symbolisieren, wird auf die Komponenten in Abbildung 5 zurückgegriffen. Das Vorgehen mit Start und Konfiguration der Anwendung, sowie die Berechnung der Metriken und Generierung der graphischen Darstellungen werden ab Punkt 2.3 an einem Beispiel einer Softwareprüfung erläutert.

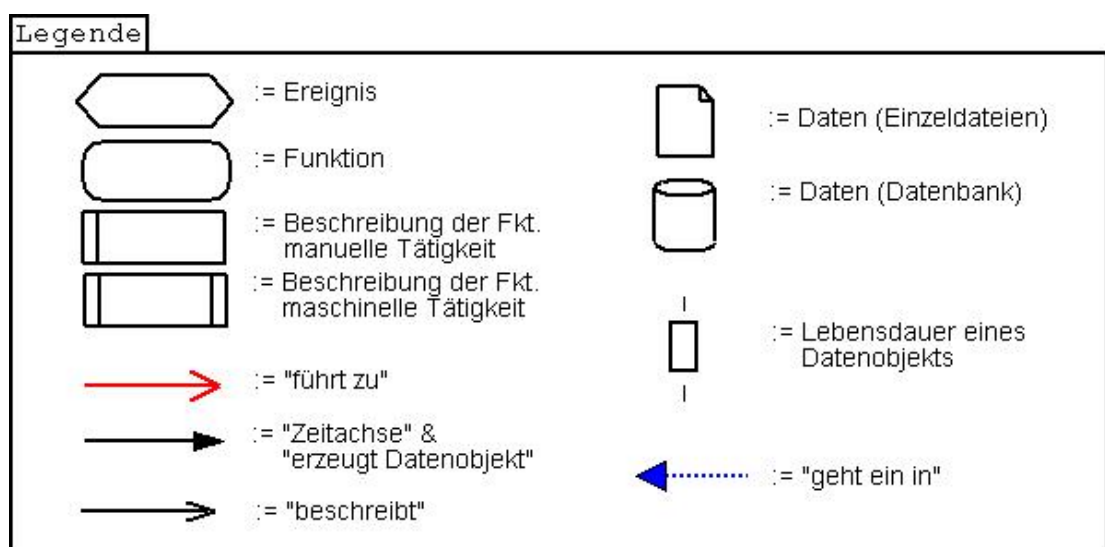


Abbildung 5 – Legende Ablaufdiagramm einer Softwareprüfung

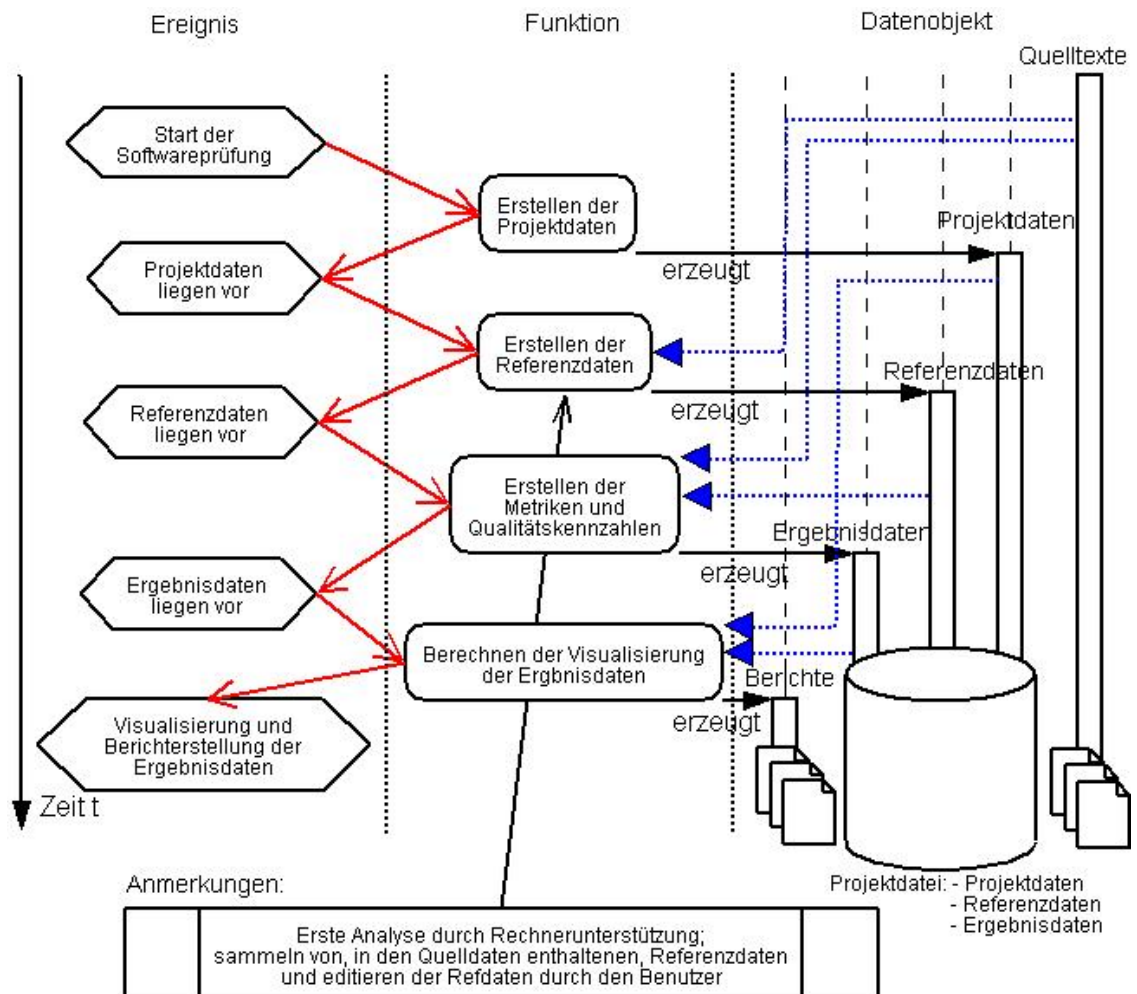


Abbildung 6 – Ablauf einer Softwareprüfung

Wird die Applikation zum ersten Mal gestartet befindet sich der Anwender im Hauptformular gemäß Abbildung 2.

2.3 Erste Schritte in der Projektverwaltung

Um eine Softwareprüfung durchzuführen, muss als erstes eine Projektdatei, für die bei der Prüfung anfallenden Daten, lokal oder auf einem Rechner im Netzwerk angelegt werden. Über „neues Projekt erzeugen“ öffnet sich ein Dialogfenster, in dem man einen Projektnamen eintragen und den Projekttyp auswählen kann.

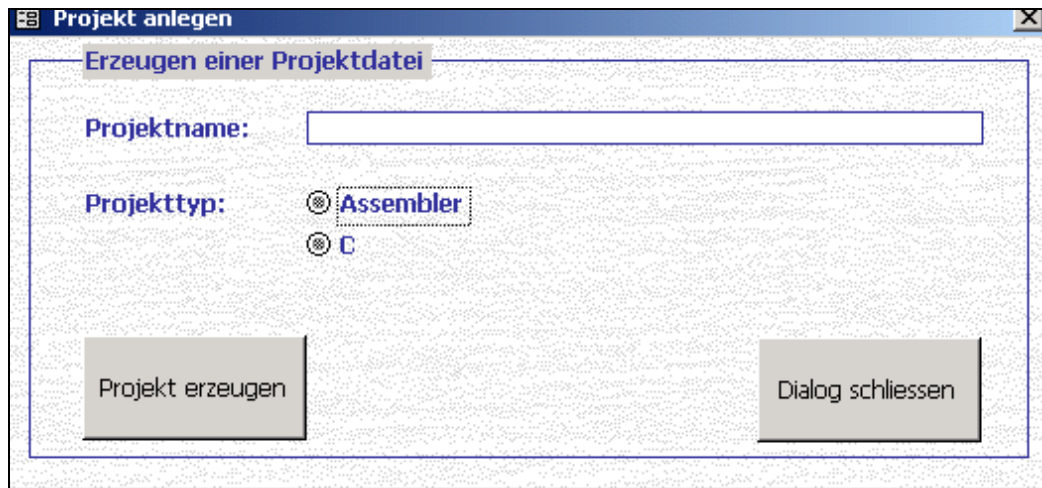


Abbildung 7 – Dialog Projekt anlegen

Der Button „Projekt erzeugen“ öffnet einen Datei – Dialog, anhand dessen der Benutzer zum Speicherort navigiert und den Dateinamen für die Projektdatei angibt.

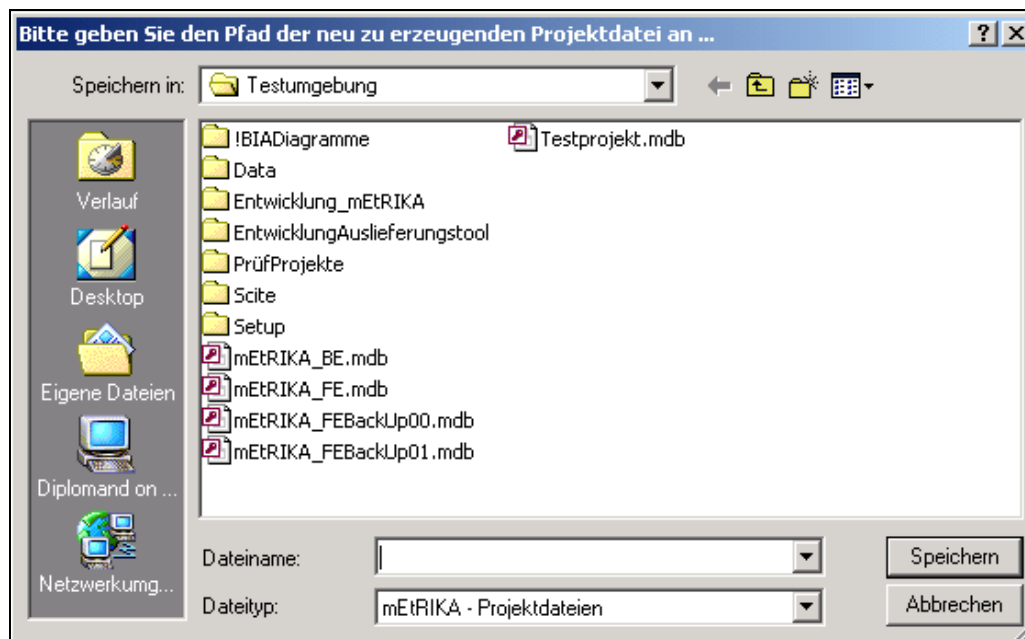


Abbildung 8 – Speichern der Projektdatei

Der anschließende Klick auf „Speichern“ legt die Projektdatei an und führt zurück zum Hauptformular.

Ist bereits ein Projekt angelegt, dass man öffnen möchte, besteht die Möglichkeit über den Button „vorhandenes Projekt in Liste aufnehmen“ einen Projektnamen für die persönliche Projektliste zu vergeben und anschließend die angelegte Projektda

tei über wiederum einen Datei – Dialog durch Auswahl und „Öffnen“ in die persönliche Projektliste aufzunehmen.

Markiert man einen Eintrag der persönlichen Projektliste durch einen Mausklick, wird eine Art „Ampelsystem“ rechts neben der Liste sichtbar. Steht die Ampel auf „grün“ existiert die Projektdatei unter dem, in der Liste hinterlegten Projektpfad. Ist dies bei einem Projekt nicht der Fall, zeigt die Ampel das rote Licht und lässt einen Button unter der Ampel erscheinen, über den der Link / Pfad zur Projektdatei aktualisiert werden kann. Dazu muss lediglich die Projektdatei über den Datei – Dialog „gesucht“ und „geöffnet“ werden.

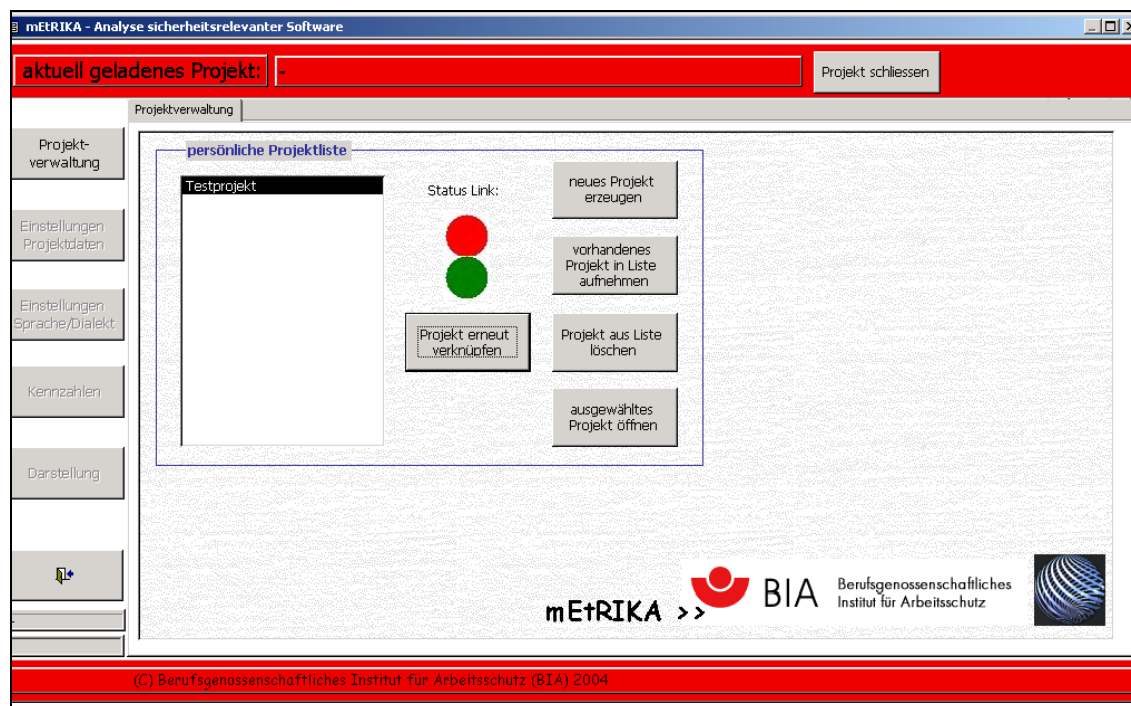
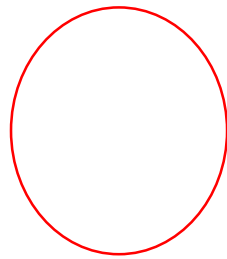


Abbildung 9 – Ampelsystem und aktualisieren des Projektpfads

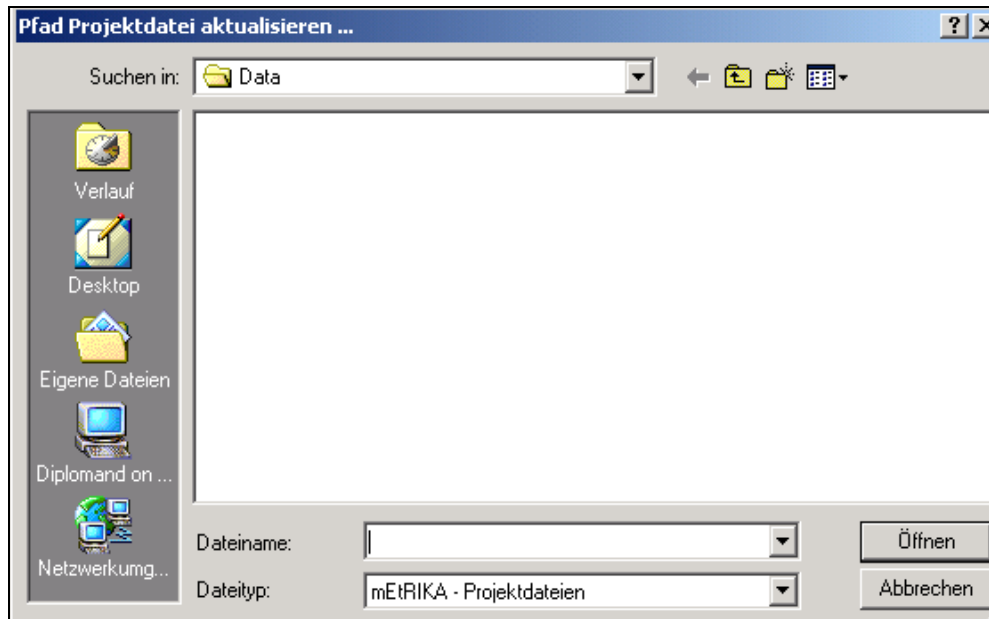


Abbildung 10 – Dateidialog zur Aktualisierung des Projektpfads

Wird der Link nicht aktualisiert, kann die Projektdatei nicht mit dem Frontend verknüpfen werden, so dass die Bearbeitung dieses Projekts nicht fortgeführt werden kann. Dieses Element lässt sich dann lediglich durch Auswahl in der Liste und anschließendem Klick auf „Projekt aus Liste löschen“ aus der Projektliste entfernen.

Ist der Zugriff auf eine ausgewählte Projektdatei möglich, können die Projektdaten über einen Doppelklick auf das Listenelement, oder durch Auswahl und anschließenden Klick auf den Button „ausgewähltes Projekt öffnen“ mit dem Frontend gelinkt werden, so dass die Verarbeitung fortgeführt werden kann. Während des „öffnen“ – Mechanismus wird automatisch von der Projektdatei eine Backup – Kopie angelegt. Das Backup wird in dem Verzeichnis, in dem auch die Projektdatei zu finden ist, erzeugt und trägt den Dateinamen NameProjektdatei“.Backup“. Sollte die Originaldatei während einer Verarbeitung beschädigt werden, lässt sie sich somit immer wieder durch Umbenennung in die Dateiendung „.mdb“ wiederherstellen.

2.4 Verwaltung und Konfiguration der Projektdaten

Handelt es sich bei dem gelinkten Projekt um ein neu angelegtes, sind eine Fülle von Projektdaten anzugeben. Sie dienen einerseits der Dokumentation des Projekts und werden an späteren Stellen Inhalt des Berichtswesens, oder sind für die weitere Verarbeitung und Analyse des Prüflings erforderlich. Bei den verschiedenen Daten

wird zwischen Projekt- und Kundendaten, projekttypspezifischen Einstellungen, die abhängig von dem zu untersuchenden Prüfling sind und den zu untersuchenden Quelldateien unterschieden.



The screenshot displays the 'mEIRIKA - Analyse sicherheitsrelevanter Software' application window. At the top, a red bar indicates the 'aktuell geladenes Projekt: Testprojekt' with a 'Projekt schließen' button. Below this, a sidebar on the left contains menu items: 'Projektverwaltung', 'Einstellungen Projektdaten', 'Einstellungen Sprache/Dialekt', 'Kennzahlen', and 'Darstellung'. The main area is divided into three tabs: 'Projekt- und Kundendaten' (selected), 'Projekteinstellungen (Assembler - Projekt)', and 'Quelldateien'. The 'Projekt- und Kundendaten' tab is further divided into 'Projektdaten' and 'Kundendaten' sections. The 'Projektdaten' section includes fields for 'Projektname' (Testprojekt), 'Version', 'Projektklasse' (Assembler), 'Projektdatei' (Testprojekt.mdb), 'Ansprechpartner im BIA', 'Telefonnummer', 'Projekt angelegt am' (30. Jun. 04), and 'Projektpfad' (D:\Testumgebung\Testprojekt.mdb). The 'Kundendaten' section includes fields for 'Kundenname', 'Ort', 'Anschrift (Strasse, Hausnr.)', 'Ansprechpartner beim Kunden', 'PLZ' (0), and 'Telefonnummer'. A red bar at the bottom of the window contains the copyright notice: '(C) Berufsgenossenschaftliches Institut für Arbeitsschutz (BIA) 2004'.

Abbildung 11 – Formular Projekt- & Kundendaten

Die spezifischen Daten betreffen beispielsweise Kommentar einleitende Zeichen im zu analysierenden Quellcode, Zeichen, die Parameter bei Befehlen voneinander Entwicklung eines Analysewerkzeugs zur Ermittlung von Metriken und Qualitätskriterien sicherheitsrelevanter Software im Maschinenschutz

trennen, Zeichen, die zwischen einzelnen Parametern auftauchen können und weiteren Einstellungen die zur Konfiguration des Analysemoduls von mEtRIKA gemacht werden müssen.



The screenshot displays the 'mETRIKA - Analyse sicherheitsrelevanter Software' application window. At the top, a red bar indicates the 'aktuell geladenes Projekt: Testprojekt' with a 'Projekt schließen' button. Below this, a sidebar on the left contains navigation buttons: 'Projektverwaltung', 'Einstellungen Projektdaten', 'Einstellungen Sprache/Dialekt', 'Kennzahlen', and 'Darstellung'. The main area is titled 'Einstellungen Projektdaten' and contains several tabs: 'Projekt- und Kundendaten', 'Projekteinstellungen (Assembler - Projekt)', and 'Quelldateien'. The 'Projekteinstellungen (Assembler - Projekt)' tab is active, showing a 'weiterführende Projekteigenschaften' section. This section includes input fields for 'Kommentar - einleitende Zeichen:' (with a '*' in the input and a list of symbols in the dropdown), 'Trennzeichen zwischen Parametern:' (with a ';' in the input and a list of symbols in the dropdown), 'Kennzeichen Quellcode:' (with a dropdown menu showing '{Kommentar}metrika'), 'Kennzeichen Label:' (with a dropdown menu showing '{ersterAusdruckInZeile}:'), and 'Kennzeichen Komponenten:' (with a dropdown menu showing 'Start einer Komponente beim ersten {Kommentar}metrika'). A footer bar at the bottom contains the copyright notice: '(C) Berufsgenossenschaftliches Institut für Arbeitsschutz (BIA) 2004'.

Abbildung 12 – Formular projektspezifische Einstellungen

Die Einstellungen zum „Kennzeichen Quellcode“ stehen initial auf „{Kommentar}metrika“, was meint, dass der zu untersuchende Quellcode im Anschluss des ersten Kennzeichens für das Analysemodul auffindbar ist. Ist das Kommentarzeichen eines Assemblercodes ein „*“, so würde die Marke für das Analysemodul „*metrika“ lauten. Ein Label wird durch die Einstellung erkannt, dass es sich wenn es der erste

Begriff einer Zeile ist und in der gleichen Zeile dahinter ein „:“ folgt. Über die letzte Konfiguration „Kennzeichen Komponente“ lässt sich einstellen, woran eine zu untersuchende Komponente erkannt wird. Beginnt eine Komponente entsprechend dem Kennzeichen Quellcode mit „{Kommentarzeichen}metrika“, so bilden alle Zeichen der Zeile die hinter Marke folgen den Namen der Komponente. Wird die Einstellung „Jede Quelldatei entspricht einer Komponente“ ausgewählt, bilden nicht die einzelnen Marken die zu untersuchenden Komponenten, sondern die einzelnen Quelldateien.

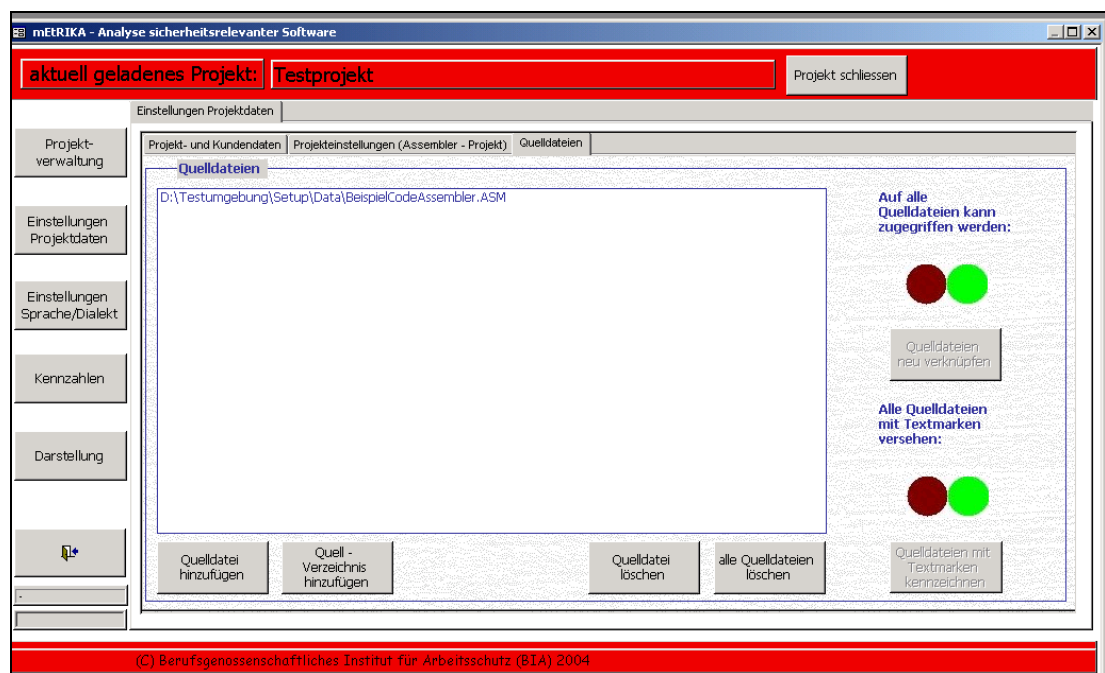


Abbildung 13 – Formular Quelldateien

Um eine Softwareprüfung durchführend zu können, müssen die zu analysierenden Quelldateien in einer Liste angelegt werden. Über „Quelldatei hinzufügen“ können einzelne Quelltexte der Liste hinzugefügt werden. Handelt es sich bei einem Prüfprojekt um ein Verzeichnis mit vielen Quelldateien, können diese der Liste auch über die Schaltfläche „Quellverzeichnis hinzufügen“ und der Auswahl des Verzeichnis

ses hinzugefügt werden. Einzelne, oder alle Quelldateien können über deren Markierung und die entsprechenden Buttons aus der Liste entfernt werden.

Um ein Beispiel einer Softwareuntersuchung durchführen zu können, existiert eine Beispielfiledatei eines Assemblerdialekts im Programmverzeichnis von mEtRIKA unter „.\Data\BeispielCodeAssembler.ASM“. Um das Beispiel exemplarisch durchzuarbeiten, wird diese Quelldatei nun als Prüfgegenstand eingestellt und daher in die Liste der Quelldateien aufgenommen.

Entsprechend dem Ampelsystem aus der Projektverwaltung, wird in der Verwaltung der Quelldatei deren Verfügbarkeit geprüft und der Status angezeigt. Werden einzelne Quelldateien nicht mehr unter dem eingestellten Pfad gefunden, lässt sich eine Liste derer über den Button „Quelldateien neu verknüpfen“ generieren, so dass die Pfadinformationen leicht aktualisiert werden können.

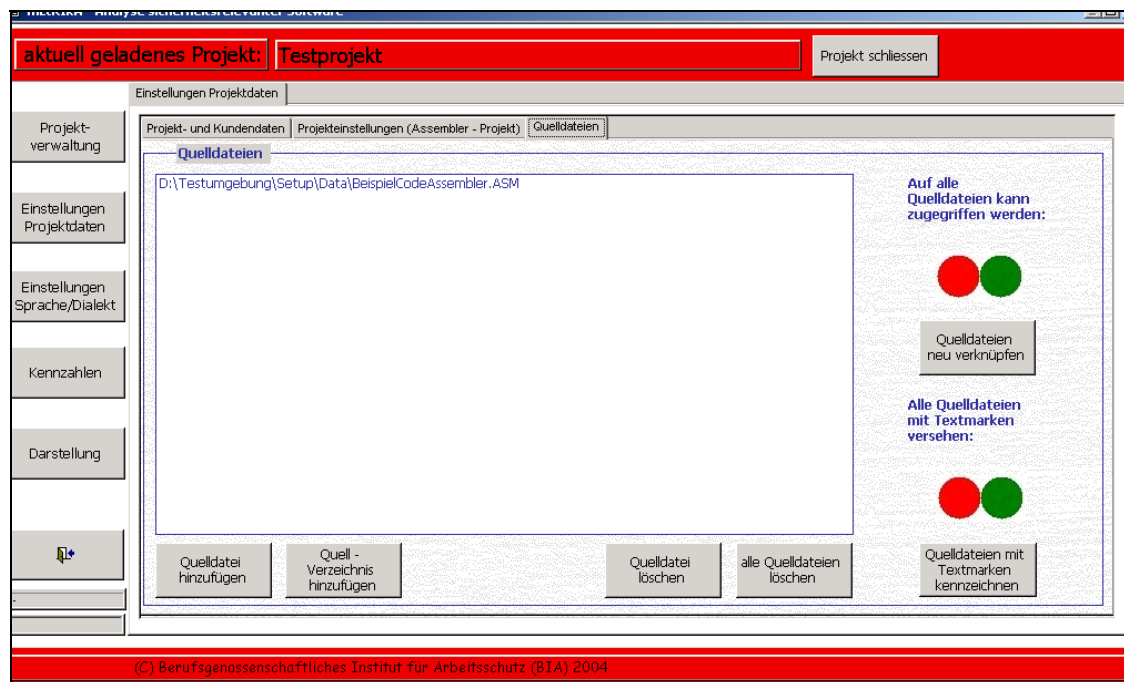
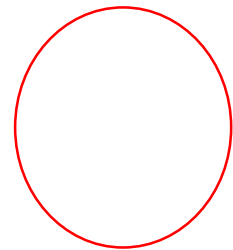


Abbildung 14 – Quelldateien Ampelsystem

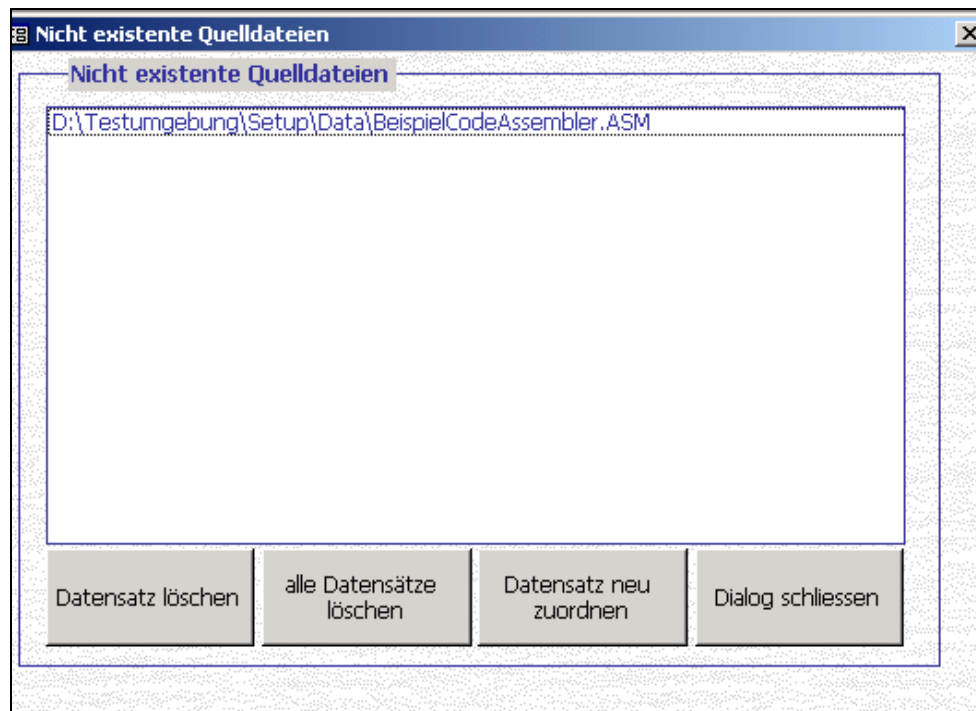


Abbildung 15 – Dialog „nicht existierende“ Quelldateien neu verknüpfen

Das zweite Ampelsystem zeigt an, ob alle, in der Liste angelegten Quelldateien, mit der mEtRIKA – Marke versehen sind, so dass das „Kennzeichen Quellcode“ in jeder zu untersuchenden Datei zu finden ist. Über den Button „Quelldateien mit Textmarken kennzeichnen“ kann eine Liste der Elemente generiert werden, für die dies noch nicht gilt.

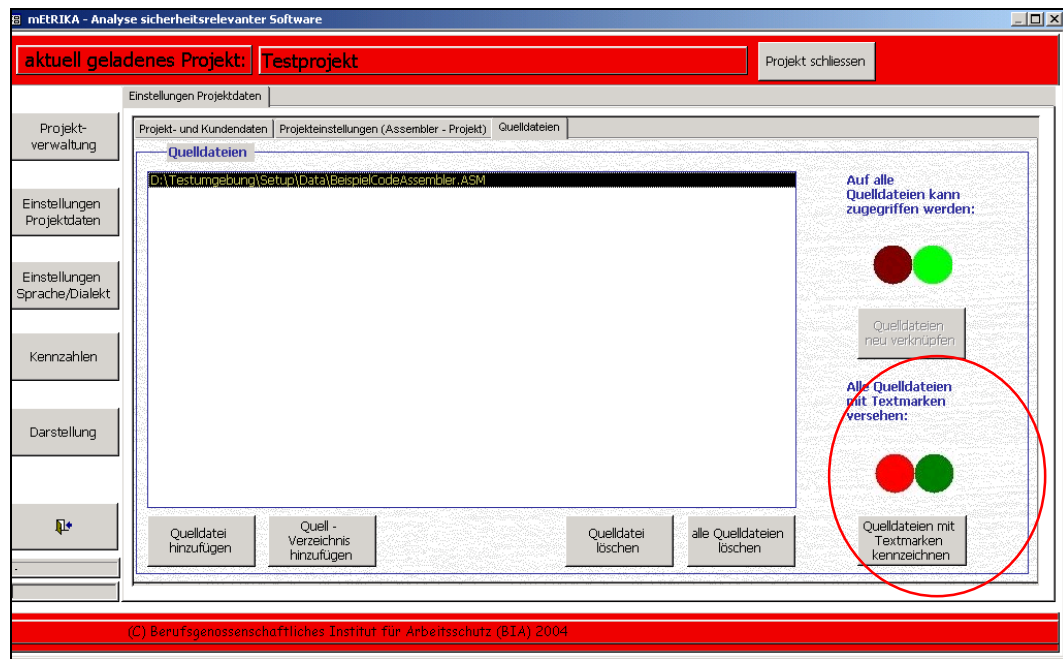


Abbildung 16 – Quelldateien ohne Textmarken

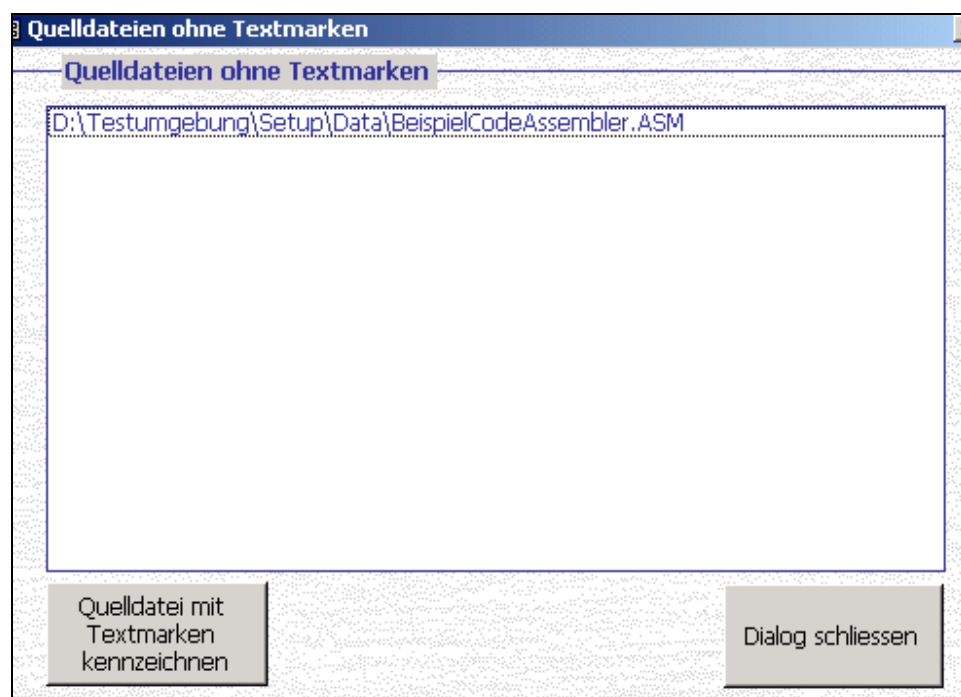


Abbildung 17 – Dialog Quelldateien „mit Textmarken versehen“

Über einen Doppelklick auf ein Listenelement der Quelldateien – Liste wird dieses mit dem, bei der mETRIKA – Installation enthaltenen, SciTE – Editor geöffnet. So kann der Anwender effizient die Quelldatei mit Textmarken versehen, beziehungsweise bei Bedarf anderweitig editieren.

2.5 Einstellungen zu Programmiersprache / Dialekt der zu untersuchenden Software und Start der Berechnung

Wurden im vorigen Schritt die Quelldateien und sonstigen Projektdaten angelegt und eingestellt, kann die weitere Verarbeitung beginnen. Um die Kennzahlen und Metriken auf Grundlage der Quelltexte berechnen zu können müssen, im nächsten Prozess die Einstellungen zur verwendeten Programmiersprache geleistet werden. Dies wird durch eine Automatisierung dem Benutzer erleichtert.

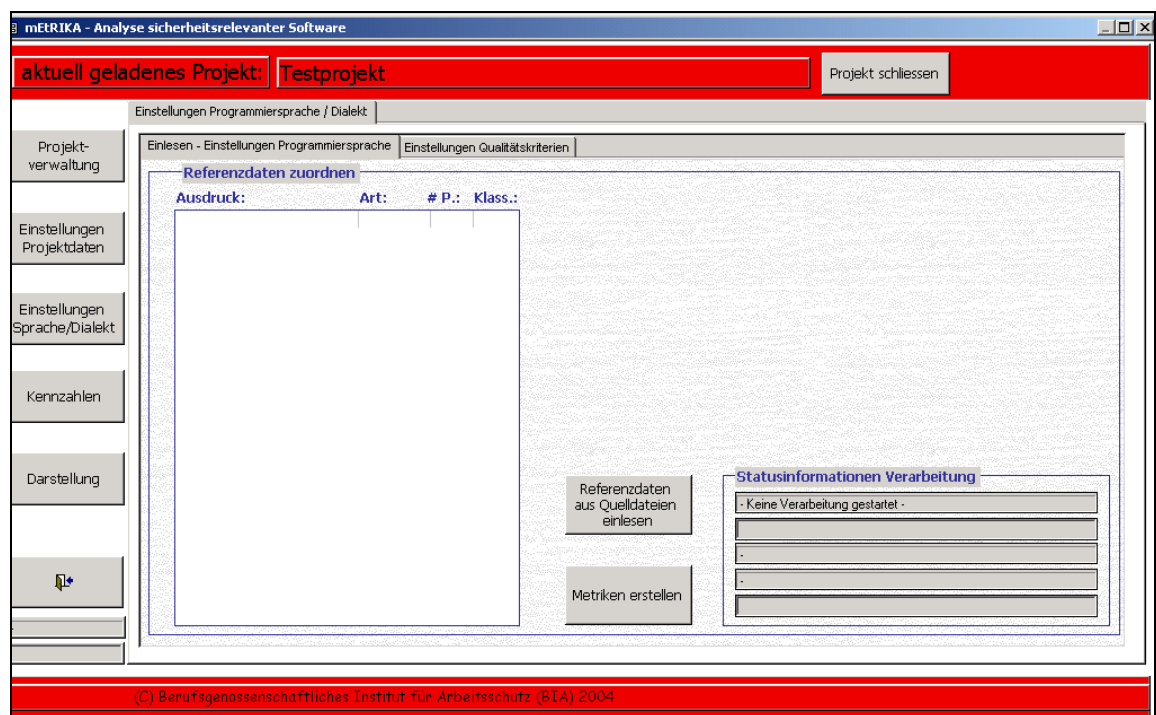


Abbildung 18 – Formular Einstellungen Programmiersprache – Referenzdaten einlesen

Über den Button „Referenzdaten aus Quelldateien einlesen“ werden die zuvor eingestellten Quelldateien ein erstes Mal analysiert, wobei in den Texten vorkommende Ausdrücke gesammelt und mit Ende der Automatik in der Liste der Referenzdaten alphabetisch dargestellt werden. Die Liste liefert an dieser Stelle eine Übersicht an enthalten Ausdrücken. Bei der ersten Analyse ist eine Art Vorhersage enthalten, Entwicklung eines Analysewerkzeugs zur Ermittlung von Metriken und Qualitätskriterien sicherheitsrelevanter Software im Maschinenschutz

welche Ausdrücke als Label, Befehl oder Parameter erkennt und entsprechend klassifiziert. Ein Doppelklick auf einen Listeneintrag öffnet den SciTE – Editor mit dem Quelltext und der Position des Listeneintrags in dem Quelltext. Dabei ist auffällig, dass die Ausdrücke jetzt auch von dem Editor „ge – highlight – et“ werden.

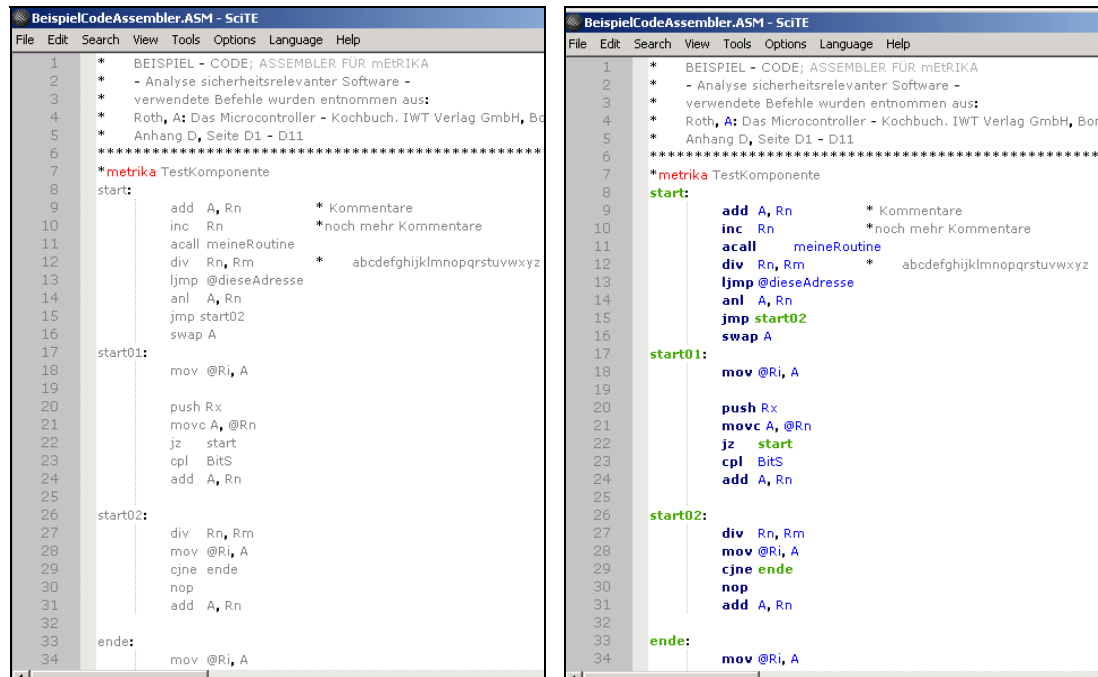


Abbildung 19 – SciTE – Editor nicht "ge-high-light-et" & "ge-high-light-et"

Da die Automatik der Referenzdatenbestimmung die gesammelten Ausdrücke nach ihrer Klassifikation in die Konfiguration des SciTE – Editors schreibt, werden die als Referenzdaten angelegten Ausdrücke zukünftig in dem Editor markiert dargestellt und sind so intuitiv und schnell für den Gutachter erkennbar.

Ebenso mit der ersten Generierung der Sprachdaten wird ein neuer Reiter „Bearbeiten – Einstellungen Programmiersprache“ sichtbar. Dieser Reiter ermöglicht die manuelle Änderung und Verwaltung der Referenzdaten. Eine Liste der angelegten Referenzdaten ermöglicht die übersichtliche Darstellung angelegter Referenzdaten. Filter- und Sortierfunktionen der Liste erleichtern das manövrieren innerhalb der Datensätze. Wird über einen Mausklick ein Datensatz der Liste markiert, filtert das Formular nach diesem, so dass rechts neben der Liste der Datensatz sichtbar und editierbar wird. Ein Doppelklick auf die Liste erlaubt dem Anwender wiederum den Ausdruck im Kontext des Quelltextes über den SciTE – Editor zu betrachten. Diese

Funktionalität soll dem Benutzer die Klassifizierung und Konfiguration des Ausdrucks erleichtern.

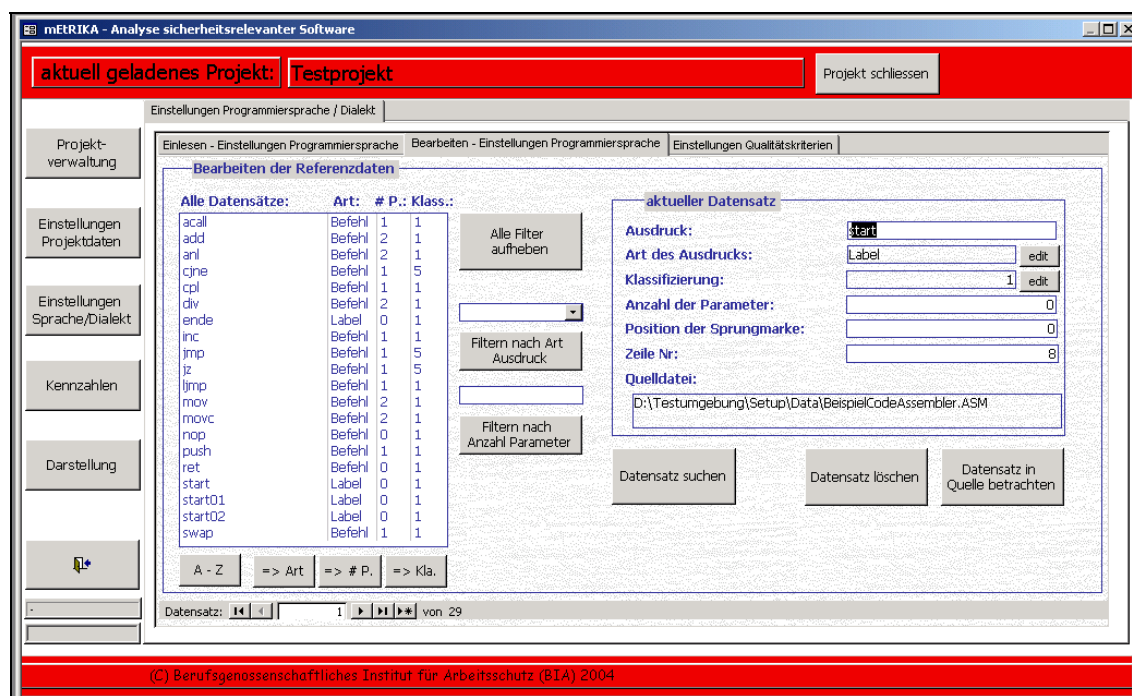


Abbildung 20 – Formular Einstellungen Programmiersprache manuell bearbeiten

Via „edit“ bei „Art des Ausdrucks“ lässt sich der Ausdruck als „Befehl“, „Label“ oder „Parameter“ mit einem Doppelklick auf den Listeneintrag einstellen. „edit“ der „Klas

sifizierung“ ermöglicht eine Einstellung des Ausdrucks nach Kategorien, die die Analyse und somit die Bewertung des Ausdrucks bei der Analyse beeinflussen. Einstellungen sind hierbei „Instruktion ignorieren“, „Instruktion analysieren“ oder, bei Sprungbefehlen beispielsweise „Sprung – von Bedingung abhängig“ oder „unbedingter Sprung“. Ist ein Befehl als Sprung klassifiziert, ist es nötig eine Position Sprungmarke > null zu vergeben, da über diese Einstellung für das Analysemodul ersichtlich wird, an welche Stelle ein Sprung bedingt oder unbedingt verzweigt.

Um nach Generierung und Berechnung der Metriken, zu diesen, entsprechend des Qualitätsmodells, Qualitätskriterien berechnen zu können, werden unter dem Reiter „Qualitätskriterien“ die Grenzwerte der Einzelmetriken konfiguriert. Änderungen wirken sich unmittelbar auf die Darstellung der Metriken und die Berechnung der globalen Kriterien aus.

mETRIKA - Analyse sicherheitsrelevanter Software

aktuell geladenes Projekt: Testprojekt Projekt schliessen

Einstellungen Programmiersprache / Dialekt

Einlesen - Einstellungen Programmiersprache | Bearbeiten - Einstellungen Programmiersprache | Einstellungen Qualitätskriterien

Einstellen der Grenzwerte für die Metriken

Halstead | McCabe | Krell

# Programmzeilen:	Obergr.:	50	Information Komponente:	Obergr.:	1255,98
	Untergr.:	1		Untergr.:	4,75
Frequenz Kommentare:	Obergr.:	2	Programm Level:	Obergr.:	1
	Untergr.:	0,5		Untergr.:	0,0277
Programmlänge:	Obergr.:	250	Programm Komplexität:	Obergr.:	36
	Untergr.:	2		Untergr.:	1
Frequenz Vokabular:	Obergr.:	4	Aufwand:	Obergr.:	73172,68
	Untergr.:	1		Untergr.:	4,75
Grösser der Instruktionen:	Obergr.:	5	Fehlerwahrscheinlichkeit:	Obergr.:	0,58
	Untergr.:	2		Untergr.:	0

(C) Berufsgenossenschaftliches Institut für Arbeitsschutz (BIA) 2004

Abbildung 21 – Formular Einstellungen Qualitätskriterien

Wurden alle wesentlichen Einstellungen gemacht, kann die Berechnung der Metriken über den Button „Metriken erstellen“ im Reiter „Einlesen – Einstellungen Programmiersprache“ angestoßen werden.

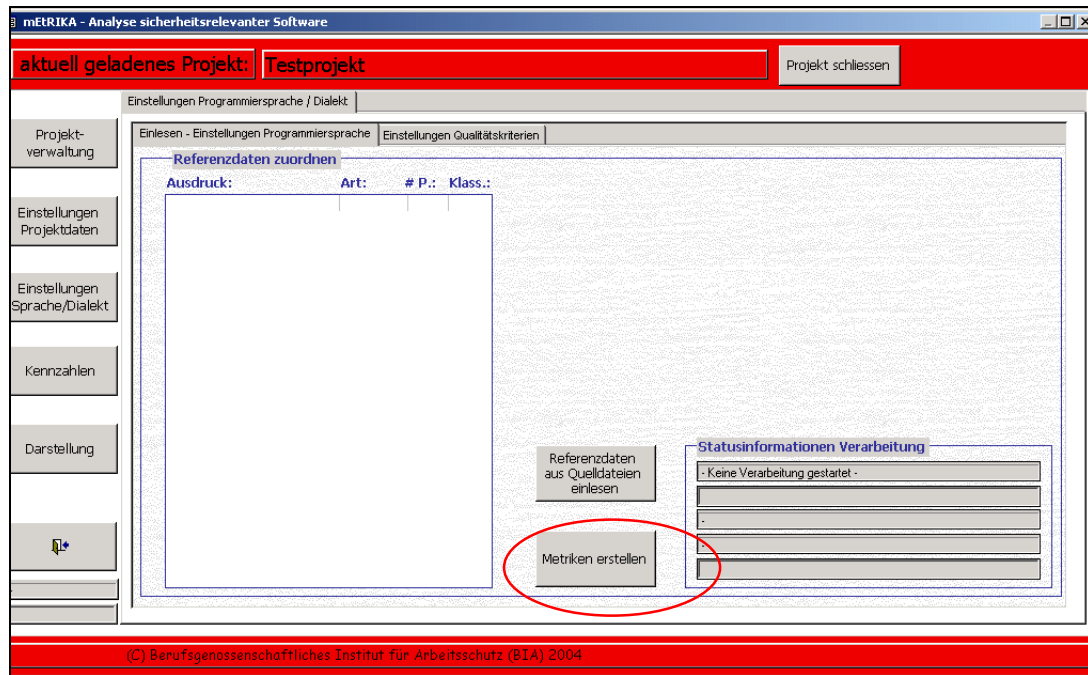


Abbildung 22 – Formular Einstellungen Programmiersprache – Metriken erstellen

Die Berechnung der Metriken kann, abhängig vom Umfang der zu Grunde liegenden Quelldateien, mehrere Stunden dauern. Ein Test auf Basis einer Assembler – Quelldatei, mit ca. 5.800 Zeilen Quellcode und ca. 200 Komponenten benötigte ca. 2,75 Stunden zur Berechnung der Metriken. Soll an dem Arbeitsplatz während der Berechnung über weitere Prozesse gearbeitet werden, ist zu empfehlen, die Prozesspriorität der Berechnung herabzusetzen. Dabei ist über den Windows – Taskmanager => Reiter „Prozesse“, der Prozess „MSACCESS.EXE“ auszuwählen und dessen Priorität über das Kontextmenü der rechten Maustaste herabzusetzen.

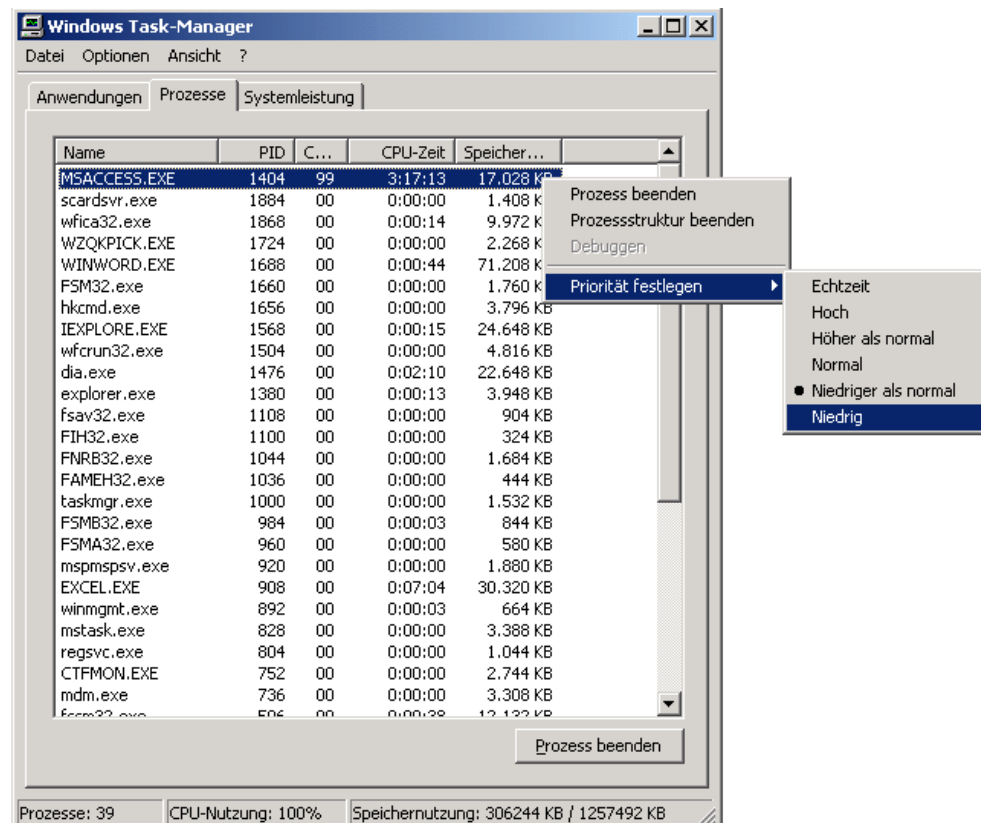


Abbildung 23 – Windows – Taskmanager: Prozesspriorität herabsetzen

Auf diese Weise kann gewährleistet werden, dass ausreichend Prozessorleistung zur Bearbeitung weiterer Prozesse zur Verfügung steht.

2.6 Berechnete Metriken und weitere Kennzahlen

Wurden alle Metriken erfolgreich berechnet, wechselt die Verarbeitung aus dem Unterformular für die „Einstellungen Sprache / Dialekt“ automatisch in die Ansicht unter der Rubrik „Kennzahlen“.

The screenshot displays the mERIKA software interface for analyzing safety-relevant software. The title bar reads "mERIKA - Analyse sicherheitsrelevanter Software". The main window has a red header bar with the text "aktuell geladenes Projekt: Testprojekt" and a "Projekt schließen" button. Below the header, there are tabs for "Kennzahlen" and "Qualitätskriterien". The "Kennzahlen" tab is active, showing a list of components on the left and a table of metrics on the right. The selected component is "TestKomponente" with 8 lines. The metrics table includes columns for various metrics and their values. The bottom of the window shows a status bar with "Datensatz: 1 von 1" and a copyright notice for BIA 2004.

Metriken - Halstead	Metriken - McCabe	Metriken - Krell	Komponenten "geblockt"	Operatoren / Operanden
# Operatoren (N1):	33			Größe der Instruktionen: 2,5714
# Operanden (N2):	21			Programmvolumen: 262,331
# Versch. Operatoren:	13			Potentielles Volumen: 104,0419
# Versch. Operanden:	16			Programm Level: 0,3966
# Kommentarzeilen:	3			Schätzwert für Prog. Level: 0,1172
# Programmzeilen:	21			Information Komponente: 30,7494
Frequenz Komment.:	0,1429			Programm-Komplexität: 2,5214
Programmlänge:	54			Aufwand: 661,4405
# Vokabeln:	29			Fehlerwahrscheinlichkeit: 0,0253
Frequenz Vokabular:	1,8621			

Abbildung 24 – Formular Kennzahlen

Hier werden alle berechneten Einzelmetriken entsprechend der gerade ausgewählten Komponente angezeigt. Handelt es sich um mehrere Komponenten, lässt sich über die Auswahl eines Listenelements per Mausklick, oder über die Navigationsschaltflächen unten im Formular, zwischen den Datensätzen manövrieren. Welche Komponente gerade ausgewählt wurde, ist dem Kopf des Unterformulars zu ent

nehmen, welcher zentrale Informationen anzeigt. Ebenso wie bei der Datensatznavigation bei der manuellen Konfiguration der Referenzdaten, werden die Formulardaten bei einem einfachen Klick auf die Liste nach der Auswahl gefiltert. Um über die untenliegenden Navigationsschaltflächen zu steuern, muss zuvor der Formularfilter über einen Klick auf den Button „Alle Filter aufheben“ entfernt werden. Ein Doppelklick auf eine Komponente in der Liste ermöglicht, die unmittelbare Verknüpfung der berechneten Werte zu den Quelltexten durch öffnen der Komponente anhand des SciTE – Editors. Rechts neben der Liste werden die generierten Einzelmetriken dargestellt. Gemäß den Grenzwerteinstellungen, die unter „Einstellungen Sprache / Dialekt“ gemacht werden können, werden die Metriken bei Unterschreitung des Schwellwerts blau markiert, hingegen Überschreitungen rot angezeigt werden.

Die Reiter „Komponenten geblockt“ und „Operatoren / Operanden“ zeigen weiterführende Ergebnisse der Berechnungen. Die geblockten Komponenten rühren aus der Berechnung der McCabe – Metriken, die die Programmstruktur untersuchen. Dabei entspricht der Quellcode einer Komponente der geblockten, nur dass der Gesamtcode hinsichtlich weniger relevanter Aspekte komprimiert dargestellt wird. Über einen Doppelklick auf die Listen, ist der SciTE – Editor wiederum mit der relevanten Quelltextstelle zu öffnen. Der Reiter „Operatoren / Operanden“ zeigt, ebenso wie der zuvor geschilderte, fortführende Ergebnisse aus der Berechnung der Metriken. Operanden und Operatoren sind Zwischenresultate der Halstead – Codemetriken und bilden eine Grundlage für die unter „Metriken – Halstead“ zu findenden Kennzahlen.

mERIKA - Analyse sicherheitsrelevanter Software

aktuell geladenes Projekt: **Testprojekt** Projekt schließen

Kennzahlen

ausgewählte Komponente: TestKomponente Quelldatei: D:\Testumgebung\Setup\Data\BeispielCodeAssembler.ASM Metrik erstellt am 30.06.2004

Komponenten Qualitätskriterien

Testbarkeit			
	von	bis	Projekt
Akzeptiert:	99	100	99
zu strukturieren:	60	98	
zu kürzen:	30	59	
neu schreiben:	0	29	

Lesbarkeit			
	von	bis	Projekt
Akzeptiert:	100	100	50
zu dokumentieren:	30	99	
zu inspizieren:	0	29	

Selbstbeschreibung			
	von	bis	Projekt
Akzeptiert:	100	100	100
zu dokumentieren:	0	99	

globale Qualität			
	von	bis	Projekt
Akzeptiert:	100	100	59,6
zu dokumentieren:	60	99	
zu inspizieren:	50	59	
zu testen:	30	49	
neu schreiben:	0	29	

Datensatz: 1 von 1

(C) Berufsgenossenschaftliches Institut für Arbeitsschutz (BIA) 2004

Abbildung 25 – Formular Qualitätskriterien

Wie eine ausgewählte Komponente bei Berechnung der Qualitätskriterien abgeschnitten hat, lässt sich über den Reiter „Qualitätskriterien“ feststellen. Dabei werden die erreichten Werte durch Hervorhebung der erreichten Klasse hervorgehoben.

2.7 Darstellungsfunktionen

Mit der erfolgreichen Generierung der Metriken und Qualitätskriterien lassen sich letztendlich die Darstellungsfunktionen unter der Rubrik „Darstellung“ uneingeschränkt verwenden. Um nicht nur die Darstellung in Zahlenwerte für Übersichten verwenden zu müssen, bietet mEtRIKA die Darstellung der Programmstruktur als Graphen.

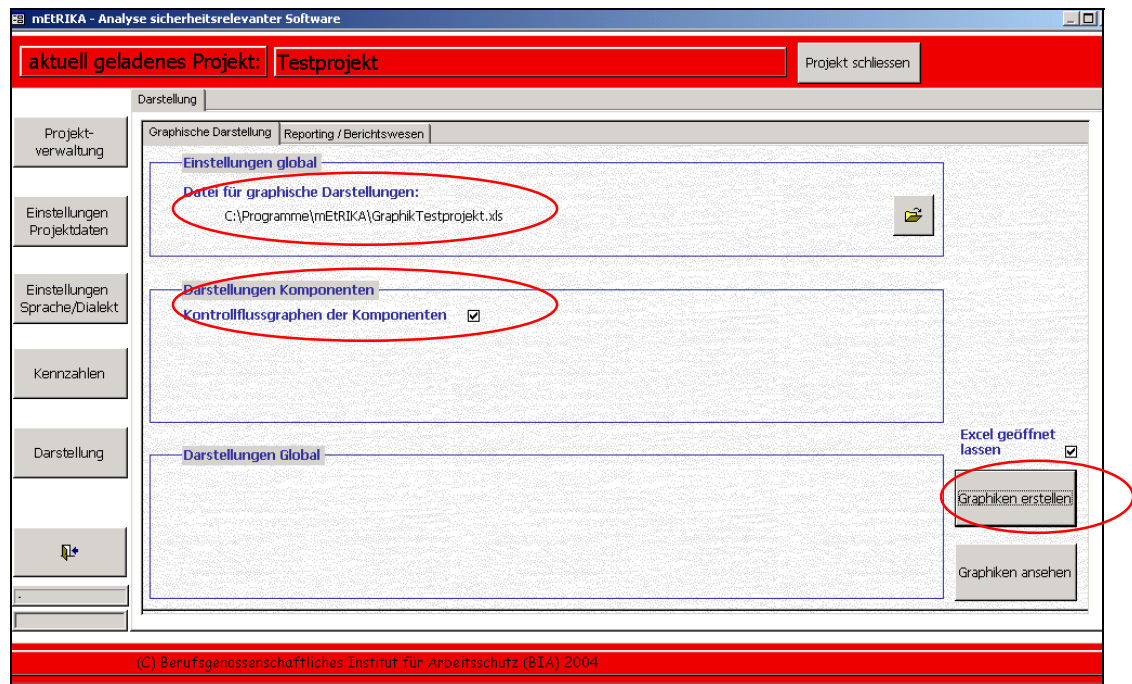


Abbildung 26 – Formular Darstellung

Welche Darstellungen generiert werden sollen, ist über die Einstellungen im Formular „graphische Darstellung“ zu steuern. Zentrale Einstellungen betreffen die Excel-datei, in die die Graphiken erzeugt werden sollen. Über den „Verzeichnis – Button“ ist der Pfad für die zu generierende Exceldatei anzupassen. Welche Visualisierungen erzeugt werden sollen, ist über Kontrollkästchen auszuwählen. Ist mindestens ein Punkt ausgewählt kann die Verarbeitung durch Klick auf die Schaltfläche „Graphiken erstellen“ gestartet werden. Mit dem darüber befindlichen Kontrollkästchen lässt sich steuern, ob die Excel – Applikation nach der Generierung geöffnet bleiben soll. Ist die Exceldatei erzeugt und die Darstellung gerade nicht sichtbar, lässt sich die Exceldatei durch Ausführen des Buttons „Graphiken ansehen“ öffnen.

Die Dauer, die mEtRIKA zum Generieren der Excelberichte benötigt, ist wie die Dauer der Berechnung der Metriken, abhängig vom Umfang der zu untersuchenden Quelltexte. Handelt es sich um mehrere Tausend Zeilen Quellcode und mehrere hundert Komponenten, ist für die Berechnung der Excelberichte ein größerer Zeit-umfang einzukalkulieren. Um in der Bearbeitung eines anderen Prozesses fortzuschreiten, kann das Fenster dieses geöffnet werden. Die Generierung der Excelbe-richte wird weiterhin über den Access – Prozess und die Excel – Arbeitsmappe fort-geführt.

Der Excelbericht lässt sich nach seinen Blättern in Teilberichte aufteilen. Entsprechend enthält jeder Teilbericht die Informationen und Werte einer Darstellung.

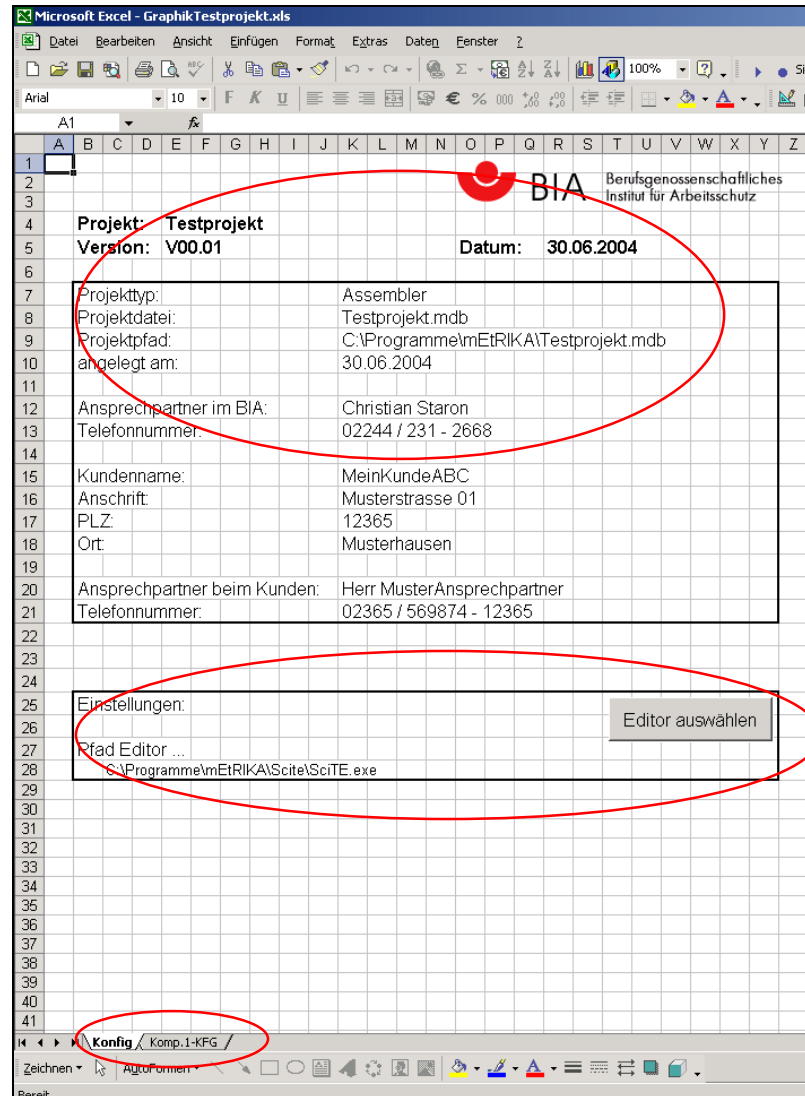


Abbildung 27 – Beispiel eines Excelberichts – Deckblatt

Mit Generierung des Berichts werden das Deckblatt und die zuvor in der Anwendung ausgewählten Teilberichte zu jeder im Projekt enthaltenen Komponente als Excelblatt erzeugt und abgespeichert. Die Exceldatei gleicht nach der Generierung einer Sammlung von Berichten, ähnlich einer Arbeitsmappe.

Das Deckblatt, welches die Bezeichnung „Konfig“ trägt, enthält allgemeine Projektdaten und darüber hinaus zusätzliche Angaben zur Konfiguration der Excelmakros, die als VBA – Code hinter der Darstellung im Excelbericht liegen. Über den Button

„Editor auswählen“ soll der SciTE – Editor ausgewählt werden, der zur Codedarstellung aus den Kontrollflussgraphen benötigt wird.

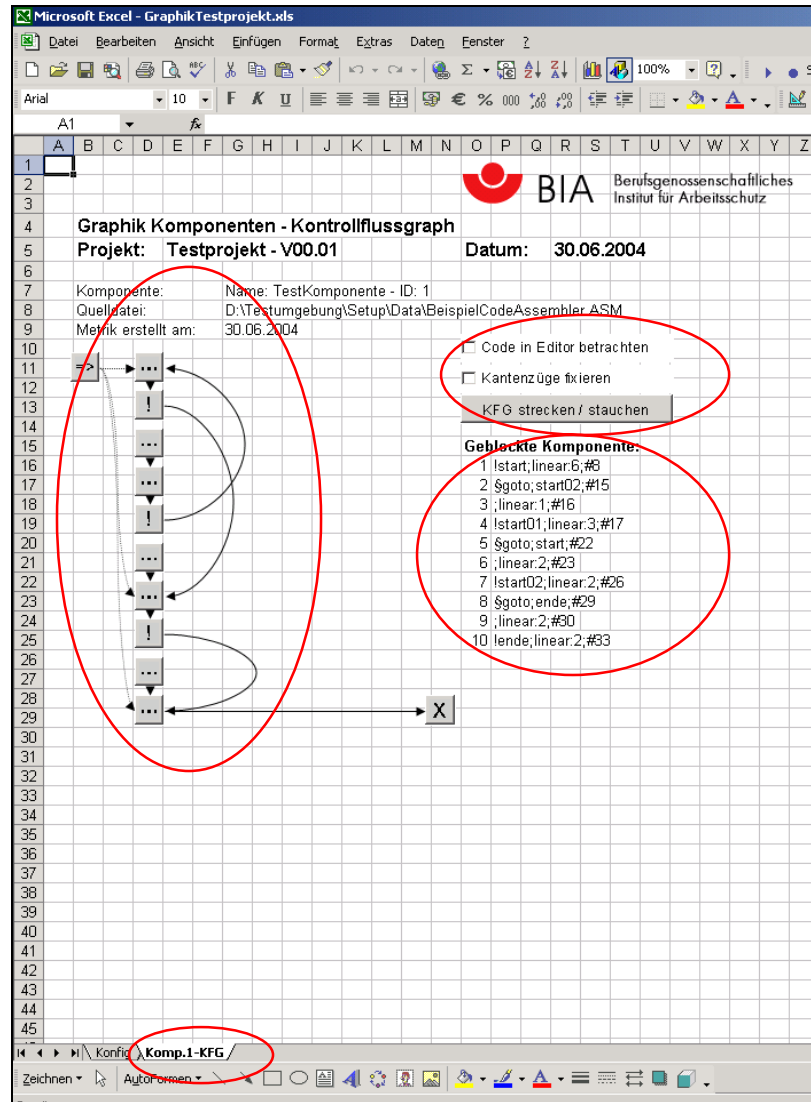


Abbildung 28 – Beispiel eines Excelberichts – Kontrollflussgraph einer Komponente

Der Bericht eines Kontrollflussgraphen (KFG) stellt im Kopf die allgemeinen Daten, auf der linken Seite den KFG selbst und rechts daneben, den Pseudocode der Komponente in geblockter Form dar. Über dem Pseudocode befinden sich weitere Elemente, die die Funktionalitäten des Berichts erweitern. Der KFG bildet den Pseudocode anhand von Symbolen ab. Ihre Bedeutung ist:

- „=>“ – Startknoten / Eingangsknoten in die Komponente
- „...“ – linearer Code innerhalb der Komponente; entspricht den Teilen

„...linear:...“ in der geblockten Komponente

- „!“ – entspricht einem unbedingten Sprung im Code der Komponente; als „§goto“ im Pseudocode zu finden
- „?“ – gemäß dem „!“ als unbedingter Sprung, entspricht es einem bedingten Sprung; im Pseudocode wird es daher als „§if“ gekennzeichnet
- „X“ – Endknoten / Aussprünge aus der Komponente

Die Kanten zwischen den einzelnen Knoten gleichen dem Programmfluss, der durch die Codeverarbeitung abgelaufen werden kann. Diese Darstellung ermöglicht eine abstrahierte Darstellung vom eigentlichen Programmcode.

Durch die Verwendung von Microsoft Excel ergeben sich eine Vielzahl von Vorteilen zur Nutzung der Berichte. Über den Einsatz gängiger Büroanwendungen lassen sich die Reports leicht via Kommunikationsinfrastrukturen verteilen und so komfortabel, selbst mit physisch weit entfernten Kollegen, besprechen. Visual Basic for Application ermöglicht weitreichende Funktionalitäten, die eine dynamische Verwendung der Berichte ermöglichen. Bei einem KFG bedeutet dies zum Beispiel, dass beim Klick auf einen Knoten die Codestelle in einem Editor angezeigt wird, Verbindungen zwischen den Knoten und die entsprechenden Stellen im Pseudocode automatisch markiert werden können.

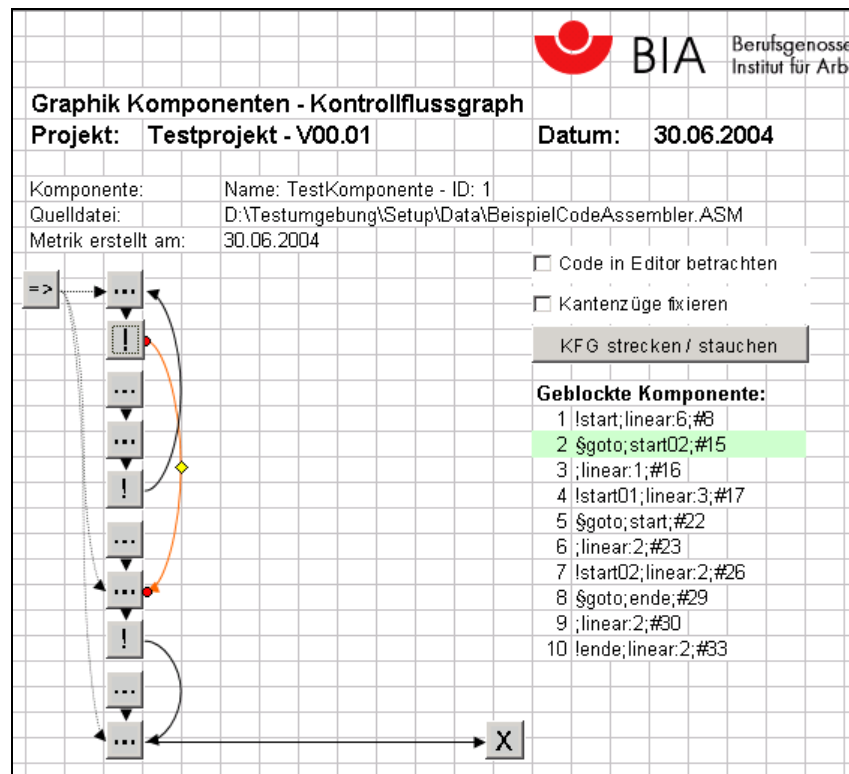


Abbildung 29 – Beispielfunktionen durch Verwendung von VBA im Berichtswesen

Erweiternd ermöglicht die Verwendung von Microsoft Excel die individuelle Berichtserstellung. Beispiele hierfür sind eigene Markierungen, beziehungsweise Darstellungen in den Berichten über in Excel enthaltene Zeichen und Steuerelement – Werkzeuge.

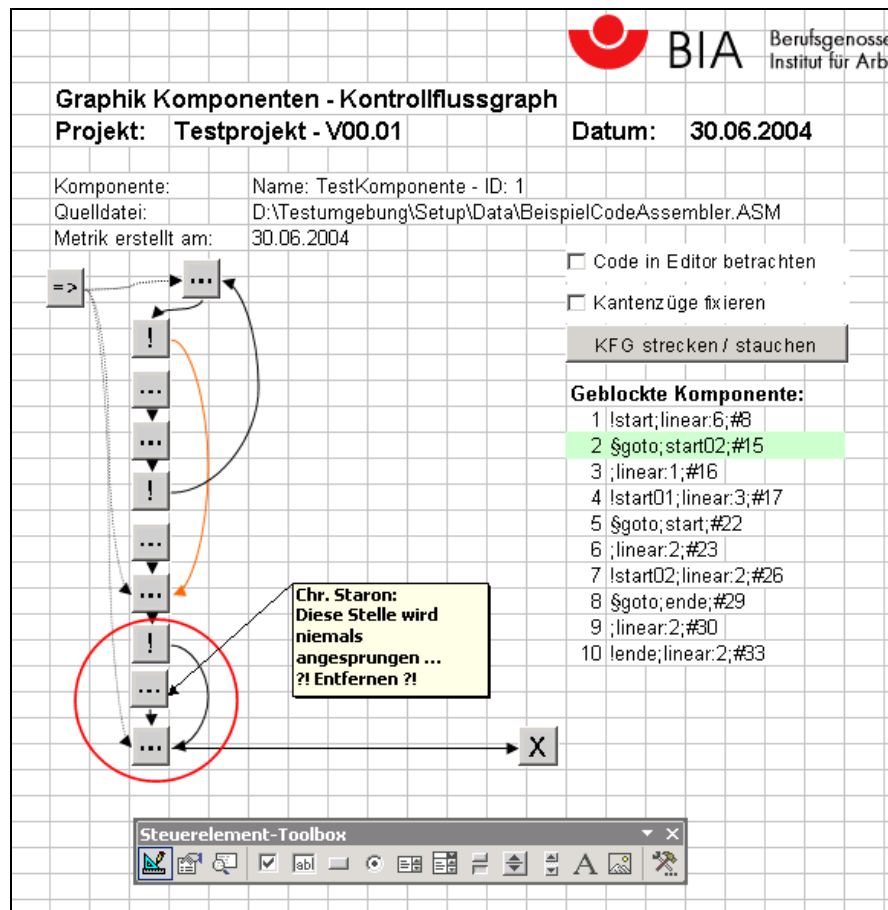


Abbildung 30 – Beispielwerkzeug Steuerelement – Toolbox, Markierung, Notiz

In der obigen Abbildung wird aufgezeigt, wie die Officewerkzeuge in den Berichten verwendet werden könnten. Durch einschalten der „Entwurfsansicht“ können Steuerelemente verschoben werden. Markierungen könnten über Zeichenfunktionen hinzugefügt werden und Notizen als Vermerke für Kollegen, oder Gedächtnisstützen in Reviews, verwendet werden.

In der bisherigen Implementierung wird die Exceldatei immer wieder erneut erzeugt. Existiert bereits eine Datei unter dem Pfad, wird sie überschrieben, so dass die vorigen Daten verloren gehen. Ziel zukünftiger Entwicklungen soll sein, ausgewählte graphische Darstellungen anhand von Excelblättern, einem bereits existierenden Excelbericht hinzuzufügen und lediglich gegebenenfalls existierende Blätter zu überschreiben. Außerdem ist ein weitreichender Ausbau der graphischen Funktionen und des Berichtswesens geplant, um darüber wichtige Repräsentationsfunktionalitäten abbilden zu können.

Erklärung

Anhand dieser eidesstattlichen Erklärung versichere ich die Bachelor – Abschlussarbeit:

„Entwicklung eines Analysewerkzeugs zur Ermittlung von Metriken und Qualitätskriterien
sicherheitsrelevanter Software im Maschinenschutz“

im Fachbereich Informatik der Fachhochschule Bonn – Rhein – Sieg, Sankt Augustin 2004,
vollständig eigens verfasst und dazu keine anderen, als die angegebenen Quellen und
Hilfsmittel verwendet zu haben.

Sankt Augustin, 27. Juli 2004

Christian Staron